U.S. Department
of Commerce

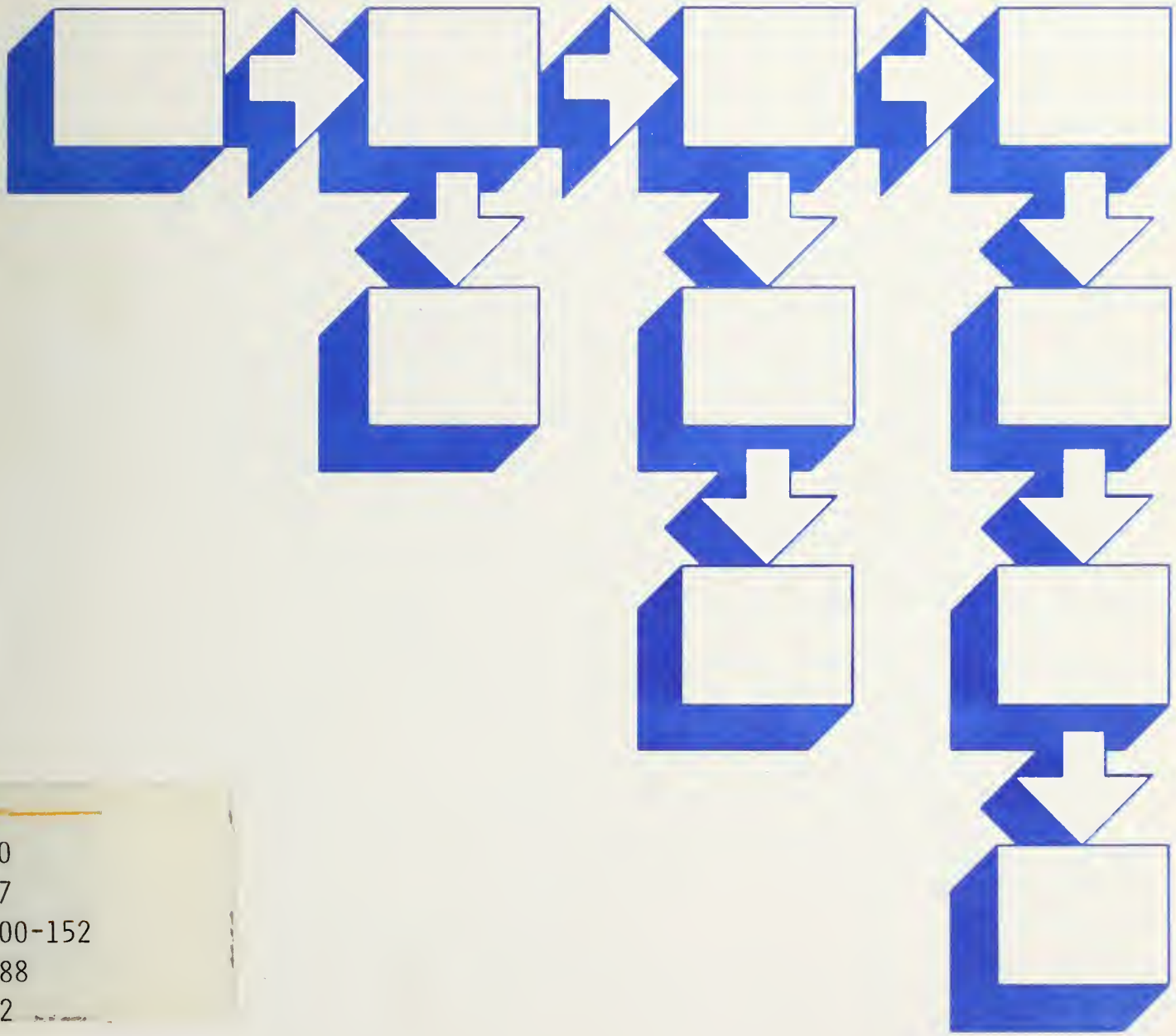
National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-152

Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning

Margaret Henderson Law



The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research to assure international competitiveness and leadership of U.S. industry, science and technology. NBS work involves development and transfer of measurements, standards and related science and technology, in support of continually improving U.S. productivity, product quality and reliability, innovation and underlying science and engineering. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, the Institute for Computer Sciences and Technology, and the Institute for Materials Science and Engineering.

The National Measurement Laboratory

Provides the national system of physical and chemical measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; provides advisory and research services to other Government agencies; conducts physical and chemical research; develops, produces, and distributes Standard Reference Materials; provides calibration services; and manages the National Standard Reference Data System. The Laboratory consists of the following centers:

- Basic Standards[2]
- Radiation Research
- Chemical Physics
- Analytical Chemistry

The National Engineering Laboratory

Provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

- Applied Mathematics
- Electronics and Electrical Engineering[2]
- Manufacturing Engineering
- Building Technology
- Fire Research
- Chemical Engineering[3]

The Institute for Computer Sciences and Technology

Conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following divisions:

- Information Systems Engineering
- Systems and Software Technology
- Computer Security
- System and Network Architecture
- Advanced Systems

The Institute for Materials Science and Engineering

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-cutting scientific themes such as nondestructive evaluation and phase diagram development; oversees Bureau-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following Divisions:

- Ceramics
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3]Located at Boulder, CO, with some elements at Gaithersburg, MD

Computer Science and Technology

NBS Special Publication 500-152

Guide to Information Resource Dictionary System Applications:

General Concepts and Strategic Systems Planning

Margaret Henderson Law

Information Systems Engineering Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

April 1988

U.S. DEPARTMENT OF COMMERCE
C. William Verity, Secretary

National Bureau of Standards
Ernest Ambler, Director

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

Library of Congress Catalog Card Number: 88-600529
National Bureau of Standards Special Publication 500-152
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-152, 145 pages (Apr. 1988)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1988

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington DC 20402

GUIDE TO INFORMATION RESOURCE DICTIONARY SYSTEM APPLICATIONS: GENERAL CONCEPTS AND STRATEGIC SYSTEMS PLANNING

Margaret Henderson Law

This guide describes the use of the Information Resource Dictionary System (IRDS) to support system development. The IRDS is the Federal Information Processing Standard (FIPS) for data dictionary systems, used to capture metadata during the system life cycle. Metadata to be stored in an IRD throughout the system life cycle is differentiated from data to be stored in a database during the system operation phase. A variety of IRD life cycle applications are described. The role of the IRDS in supporting Information Resource Management (IRM) and Data Administration is discussed. The development of the IRDS is described in terms of the evolution of data processing toward larger, more complex, integrated systems that require intensive planning and control. The preparation of a detailed standards and conventions document by an organization is recommended prior to IRD use. The procedures for developing an IRD schema and IRD metadata are described in terms of Entity-Relationship-Attribute modeling, life cycle phase partitions, and user views. Metadata integrity rules and validation procedures are discussed. The guide illustrates the use of the IRDS with an IRD application for Strategic Systems Planning. An extract of output for this IRD is presented in an appendix.

Key words: Data Administration; data dictionary system; data management; data modeling; Entity-Relationship model; E-R; Federal Information Processing Standard; FIPS; Information Resource Dictionary System; Information Resource Management; IRDS; Strategic Systems Planning.

ACKNOWLEDGMENTS

The guidance provided by Dr. Alan Goldfine, Thomasin Kirkendall, and Dr. David Jefferson during the conception of this guide is gratefully acknowledged.

Table of Contents

List of Illustrations

1.0 Introduction

1.1 Purpose

The purpose of this guide is to describe and illustrate the use of the standard Information Resource Dictionary System (IRDS). This guide is directed toward Data Administrators, Information Resource Managers, Database Administrators, and others in government and industry who are interested in the effective management of information resources.

1.2 What is an Information Resource Dictionary System?

An Information Resource Dictionary System (IRDS) is a software system that conforms to the Federal Information Processing Standard (FIPS) for data dictionary systems. An Information Resource Dictionary (IRD) is an application of the IRDS. An IRD can be used to support system life cycle information management, or it can be used for a number of other tasks. This guide describes the use of the standard IRDS in constructing IRD applications for information systems development and operations. IRDS support for the Strategic Systems Planning phase is illustrated.

1.2.1 Data Dictionary Versus Database

Data dictionary systems have been designed specifically to support the complexities of metadata. An Information Resource Dictionary, or data dictionary, is a **highly structured type of database** that can be used to design, monitor, locate, protect, and control data in information systems.

A data dictionary differs from a database, however. While a **database** holds the data **value** stored for a data item, a **data dictionary** can contain a wealth of information **about** that data item. This information about data is called **metadata.** Some examples of information about a data item that a data dictionary can support are:

- Category of the data item
- Relationships of the data item to other data items
- When and by whom it was defined
- When and by whom it was modified
- Total number of its modifications
- Description of the data item, such as its format
- Databases or files in which the data item appears
- Location of the data item in databases or files
- Set or range of valid data values permitted for a data item in the database.

Data dictionary systems, such as the IRDS, support metadata. **Metadata** is information describing the characteristics of data.

1.2.2 Data Sharing and Metadata

Information systems use programs to input, modify, and access data, which is stored either in databases or in files. While data was once considered subordinate to the programs that direct data processing, data is now recognized to play a critical role in information systems.

Data dictionary systems, such as the IRDS, support the design of efficient programs and databases for improved information systems that **share data**. An information system usually contains multiple programs that access data from multiple databases or files. When designed properly, programs within an information system **share data and data stores**, so that data is no longer the exclusive territory of one particular program. This data sharing reduces the amount of effort necessary to prepare input data within an organization. An organization no longer must reproduce similar data in different input files to be used by a variety of programs.

Data sharing among multiple programs has other benefits as well. Data sharing improves data **accuracy**, due to better data concurrency, and it improves the **efficiency** of data storage and retrieval, due to reduced redundancy. Well-designed data structures in shared databases allow programs to update and access information more easily and cost-effectively.

The design of such databases requires the identification of **metadata** describing the **structural characteristics of data** to be stored. The IRDS supports metadata description for the design of information systems that share databases and other system resources.

Metadata can be seen in the **forms** used to capture data, such as a personnel application form. The blank application form that each job applicant fills out contains **headings or questions** (e.g., last name, first name, middle initial, date of birth, place of birth, social security number, college or university, type of degree, date of degree). These headings are metadata that structure the data to be entered on the form and, if the applicant is hired, to be stored in a database.

In this context, **metadata** is the **identification and description of these form headings** in terms of format, use, and meaning. For instance, "date of birth" may require a numeric date format of "month/day/year," while "date of degree" may permit only the numeric date format of "month/year." Such a description of data format constitutes one aspect of metadata.

1.2.3 Data Versus Metadata

Data is stored in databases for use in **operational** systems. For example, your local supermarket probably uses a database that provides data for a grocery system. As a grocery shopper, you are interested in **data** for the particular items you are buying. When the checkout clerk draws one of your grocery items across the optical scanning device that reads the item's bar code, you are interested in the item name and its price, which are displayed on the cash register, and which appear on your sales receipt. The **price** that you paid for a **particular grocery item** is the data with which you, as a grocery shopper, are concerned. The grocery item's bar code, name, and price are stored as data in the database of the grocery system. The grocery system just described is an **operational database system.**

Figure 1 provides two example views of grocery system **data** in an **operational database system.** The shopper's view shows a portion of a sales receipt, and the grocery store manager's view shows a portion of a stock list. The **italicized** data items, which structure this system design, can also be considered **metadata** used to design this operational database system.

Grocery System Data

Shopper's Sales Receipt

ST2%MILK	1	GAL	1.78
PEPFBREAD	1	LB	.95
PEPEANBUT	12	OZ	.89
ROMLETTUC	1	LB	.99
STBUTMILK	1	QT	.85

Manager's Stock List

CATEGORY: DAIRY PRODUCT
PRODUCT: MILK
SUPPLIER: SEALTEST

ITEM TYPE	*ITEM SIZE*	*SALES PRICE*	*STOCK NO/DATE*	*SOLD-BY NO/DATE*	*RESTOCK NO/DATE*
REG	1 GAL	1.85	300 4/09/88	125 4/12/88	300 4/14/88
	1/2 GAL	1.10	400 4/10/88	89 4/13/88	375 4/16/88
2%	1 GAL	1.78	250 4/09/88	210 4/12/88	260 4/13/88
	1/2 GAL	1.10	300 4/10/88	180 4/13/88	300 4/13/88

Figure 1

Metadata is different from data in that it describes data in an operational system. While a grocery shopper would be interested in grocery shopping **data**, the person designing the grocery system would be interested in **metadata** used to **describe** and **structure the data.**

The grocery system designer's **metadata**, collected in a data dictionary such as an **IRD**, provides the structure for grocery data, collected in a database. In defining the information to be accessed with a bar code, the grocery system designer would want the system to indicate other types of information in addition to the item name and its current price. This additional metadata could include:

- **CATEGORY** -- describes data values such as "fresh produce," "dairy products," "canned goods," "frozen foods," "cleaning products," etc.
- **PRODUCT** -- describes data values such as "canned pumpkin," "milk," "grapefruit," "detergent," etc.
- **SUPPLIER** -- describes data values such as "Nabisco," "Sealtest," "Pepperidge Farm," etc.
- **LAST-SELL-DATE** -- describes data values such as "sell by 10/1/91," etc., printed on perishable, packaged grocery products that spoil if stored too long

In specifying these information elements and their domains of valid values, the system designer is working with metadata. For the example above, the designer will probably want to keep metadata that defines the **set of permitted values** for each of the terms above.

IRDS metadata representation is based on the **Entity-Relationship-Attribute** information modeling approach. In general, an **entity** is a thing or concept about which information is collected, such as a grocery system report called **STOCK LIST**, or an element of this report called **PRODUCT**. An entity is classified in terms of an **entity-type**, so that STOCK LIST is of entity-type **REPORT**, and PRODUCT is of entity-type **ELEMENT**.

A **relationship** describes the association between two entities, such as an association of **CONTAINS** between entity **STOCK LIST** of type REPORT and entity **PRODUCT** of type ELEMENT. A relationship is classified in terms of a **relationship-type**, which in this case is **REPORT-CONTAINS-ELEMENT**.

Attributes, classified in terms of attribute-types, can be used to describe either entities or relationships. Attributes could be used to define the set of permitted values for an ELEMENT such as PRODUCT, similar to the values listed above. Metadata such as this can be captured in an IRD. Figure 2 shows a number of entities of entity-type REPORT and ELEMENT, and indicates a number of **relationships** between the **REPORT** entities and the **ELEMENT** entities.

Grocery System Design Metadata

Entity-type:	REPORT	Entities:	STOCK-LIST SALES-RECEIPT DAILY-SALES-REPORT . . .
Entity-type:	ELEMENT	Entities:	CATEGORY PRODUCT SUPPLIER ITEM-TYPE ITEM-SIZE SALES-PRICE . . .
Relationship-type:	REPORT - CONTAINS - ELEMENT		

Relationships:

SALES-RECEIPT	REPORT - CONTAINS - ELEMENT	PRODUCT
SALES-RECEIPT	REPORT - CONTAINS - ELEMENT	ITEM-SIZE
SALES-RECEIPT	REPORT - CONTAINS - ELEMENT	SALES-PRICE
STOCK-LIST	REPORT - CONTAINS - ELEMENT	CATEGORY
STOCK-LIST	REPORT - CONTAINS - ELEMENT	PRODUCT
STOCK-LIST	REPORT - CONTAINS - ELEMENT	ITEM-TYPE
STOCK-LIST	REPORT - CONTAINS - ELEMENT	ITEM-SIZE
. . .		

Figure 2

Such relationship information is useful to the system designer in formatting reports, and will be useful later on during system maintenance, when element names and report formats change. Entities-Relationship-Attribute modeling for the IRDS is described at greater length in subsequent chapters.

A great deal of metadata must be managed during the design and development of any information system, even the relatively simple one described. This metadata should be stored in a specialized type of database, a data dictionary, such as an IRD.

1.2.4 Evolution to the Information Resource Dictionary System

Data dictionary systems have had several traditional system support roles. One of these has been to support database management systems (DBMSs). In a **DBMS support role**, a data dictionary system is coupled to a DBMS to record the data structures and structural changes occurring in databases generated by the DBMS. By keeping track of the way the data is represented in a database, a data dictionary provides information on the access, retrieval, and modification of data structures.

If one or more components of the information processing environment, **such as a DBMS**, depend on the data dictionary system for metadata, the data dictionary system is **active.** A data dictionary system is considered active with respect to a program or process if the **program or process retrieves metadata from the data dictionary system** [LEON82].

Another traditional role for a data dictionary system has been to standardize and document the **data elements** used by multiple programs within a system. Many large government systems depend on data element standardization that is supported by data dictionary systems. These **data element dictionaries** provide a means of:

- Recording information about data elements, such as access name, data type, and storage format, for data element documentation
- Reporting discrepancies in data element naming, definition, and description, to aid data element standardization
- Selecting data elements individually or in sets, for various types of analysis.

Analysts defining, updating, and retrieving data element information often use batch files to update and access the data dictionary. A data element dictionary such as this is considered **passive** if its metadata is **accessed by a person**, working in either interactive or batch form, but cannot be retrieved by a program or DBMS.

While data dictionary systems continue to support database management and data element standardization, they can also support a **wider range of applications.** Data dictionary systems are at the center of systems that support many different functions, ranging from logical database design, and aircraft engineering design, to program design languages.

Some state-of-the-art technology relies heavily on the support of data dictionary systems. The new and emerging Computer Aided Software Engineering (**CASE**) tools are often highly structured data dictionary systems that have been combined with analysis programs and graphics interfaces. Data dictionary systems are at the heart of emerging **distributed DBMS** software, which supports distributed database systems. Data dictionary systems, in various forms, are beginning to play more visible roles in supporting information management.

In the last decade, data dictionary systems have evolved so dramatically that they appear to have even surpassed their name. Other terms such as **directory**, **encyclopedia**, and **repository** have been used by vendors to designate the expanded uses of data dictionary systems. The name of the new data dictionary system standard, the Information Resource Dictionary System, reflects the evolution of this software into the information age.

1.3 Purpose of the IRDS Standard

The purpose of the FIPS IRDS standard is to provide U.S. Federal government data dictionary system users with useful, flexible, and user-friendly data dictionary system software products to support all phases of the system life cycle. The standards specify that these products must share a common set of features independent of implementation. These standardized IRDS features are specified in terms of core and optional modules that allow vendors to implement IRDS features modularly and incrementally in their products. The modular structure of the IRDS permits user organizations to acquire only needed modules.

1.3.1 IRDS Benefits

Both government and industry benefit with the incorporation of the IRDS specifications into commercial data dictionary system software products. Due to the common set of features shared by IRDS products, organizations with multiple IRDS products from different vendors can increase **user efficiency** and improve **metadata transportability** from system to system. IRDS standards are designed to improve the quality of data dictionary system software, by giving users **schema extensibility** and **life cycle support**, and by giving vendors a common basis from which to work.

1.3.2 IRDS Prototype

During the deliberation of IRDS technical specifications, NBS developed a prototype implementation of the IRDS command language to demonstrate technical IRDS concepts. The prototype uses a commercially available programming language and runs as an application of a commercially available DBMS. NBS provides the source code of the IRDS prototype to interested users, such as government agencies and software vendors.

1.4 Scope and Related Publications

This section describes the scope of this guide, and recommends related NBS publications for readers who want more information on the IRDS or system development process.

1.4.1 Scope

This guide illustrates the development of IRD applications for a standard IRDS, to help data dictionary system users understand the benefits of using a data dictionary system that is compatible with the IRDS standard. The development of IRD applications is demonstrated through a sample Strategic Systems Planning application.

Future publications are planned to cover the use of the IRDS in other life cycle phases such as Requirements Definition, Functional Specification, Logical Database Design, System Design, Physical Database Design, and System Implementation.

This guide is only intended to inform potential IRDS users of a type of system development life cycle application that the IRDS can be used to support; it is not intended to be a complete IRDS user's manual.

Chapter 2 describes the use of the IRDS in Information Resource Management and Data Administration. A number of IRD Data Administration applications are described. Chapter 3 discusses the existing and planned features of the IRDS standard. Standards and conventions necessary for an organization's responsible use of the IRDS are described in Chapter 4.

Chapter 5 provides an overview of how to construct an IRD schema appropriate for a planned IRD application. Since the IRDS provides limited predefined schema structures, users should plan to define additional schema structures for most IRD applications. This chapter illustrates the schema development process, from the user group's description of example problem statements, and development of an appropriate Entity-Relationship-Attribute model, to IRDS command language examples for schema definition. Life cycle phase definition is explained.

Chapter 6 describes the process of defining metadata to create an IRD application. This chapter demonstrates view definition within a life cycle phase. Command language examples are shown for defining entities, relationships, and attributes, and for determining view and phase status.

Chapter 7 briefly discusses the use of an IRD application through the system life cycle process. The transfer of metadata across phases is discussed.

Chapter 8 illustrates the use of the IRDS in the first phase of the life cycle, Strategic Systems Planning, showing the definition of sample problem statements and Entity-Relationship-Attribute models on which the IRD schema and metadata are based.

Chapter 8 illustrates the use of some features of the IRDS to support a number of Strategic Systems Planning activities for a fictitious corporation, the XYZ Corporation, and its subsidiary, Company X. The sample application is intended to assist users of the IRDS in defining schema extensions and constructing IRD applications.

Strategic Systems Planning results in an overview of an entire enterprise, its critical success factors, and its multiple system development or redesign projects. The analysis supported by an IRD during Strategic Systems Planning provides a foundation for the achievement of integrated information systems, designed to meet management objectives and to sustain the critical success factors defined for the enterprise.

Chapter 8 provides an example IRD **schema** definition for Strategic Systems Planning, showing command language definitions for the application. The definition of an IRD life cycle phase partition is illustrated. Commands to define entity-types, relationship-types, attribute-types and other schema descriptors are shown.

Chapter 8 also provides an example IRD **metadata** definition for Strategic Systems Planning, showing command language definitions for views within a life cycle phase, entities, relationships, and attributes.

Chapter 9 draws the conclusions of the guide. The Appendix offers an extract of the example IRD output for Strategic Systems Planning.

The guide does not address IRDS support for many life cycle activities such as prototyping, testing, operations, maintenance, or redesign. While some system development activities are mentioned, many system development activities are not addressed in this publication. The partial Strategic Systems Planning application defined in Chapter 8 is provided to illustrate IRDS concepts and techniques. The limited space of this guide allows illustration of only a partial and simplified example of IRD development for the Strategic Systems Planning phase.

The IRDS is recommended for use during the information system life cycle to support metadata definition and retrieval. The **large** amount of metadata usually generated during a **single** life cycle **phase**, and the **enormous** amount of metadata usually generated over the duration of a complete system **life cycle** necessitate that a data dictionary system like the IRDS be used to support and cross-reference metadata for information system development, operations, and maintenance activities.

1.4.2 Related Publications

For those interested in more information about the IRDS, other NBS publications on the topic include:

- A Technical Overview of the Information Resource Dictionary System (Second Edition), by Alan Goldfine and Patricia Konig, NBSIR 88-3700, National Bureau of Standards, Gaithersburg, MD, January, 1988.

- Using the Information Resource Dictionary System Command Language (Second Edition), by Alan Goldfine, NBSIR 88-3701, National Bureau of Standards, Gaithersburg,MD, January, 1988.

Other NBS publications indirectly related to use of the IRDS include:

- Guide on Logical Database Design, by Elizabeth N. Fong, Margaret W. Henderson [Law], David K. Jefferson, and Joan M. Sullivan, NBS Special Publication 500-122, National Bureau of Standards, Gaithersburg, MD, February 1985.

- Guide on Data Entity Naming Conventions, by Judith J. Newton, NBS Special Publication 500-149, National Bureau of Standards, Gaithersburg, MD, October 1987.

For additional NBS publications related to the IRDS or to systems development, see the references for [FONG86], [GOLD82], and [FISH87] listed at the end of this document.

2.0 IRDS Support for Data Administration

The IRDS represents Federal standards efforts to provide quality data dictionary system support for information management. An enterprise concerned with effective information management, processing, conversion, and communications should adopt a set of Information Resource Management (IRM) policies. IRM policies coordinate the development, operation, and maintenance of an enterprise's information systems. Within IRM, the primary organizational component responsible for information management is Data Administration. The IRDS provides many functions useful for Data Administration.

2.1 Information Systems

Data Processing has long been the dominant concept and the major organizational function for computer systems development. As an organizational function, Data Processing has emphasized programs and processes that transform data, rather than data management. In traditional Data Processing, data was collected for, stored with, and used by a particular program. Although Data Processing management became increasingly aware of the need for data sharing and data management, data sharing among multiple programs was often not implemented during system development. When data sharing was used, it was often limited to a few programs used by the same department. Data sharing was often only added on as an afterthought during expensive system maintenance or system redesign projects.

The relatively recent concept of information resource management derives from an understanding of the need to utilize data as a primary resource of the enterprise. As a result of emphasis on information resource management, some Data Processing departments now often have new names: Management Information Systems (MIS), Information Systems (IS), or IRM. The emerging shift from Data Processing to IRM is partially due to the wider use of computer systems, communications systems, and DBMSs. Computer systems growth has resulted in greater quantities of data to be managed from greater distances. Organizations now require larger and faster systems in which telecommunications and information retrieval are often as important as data processing.

Computer systems, which were once updated and accessed from a single location and utilized by one type of user, have become information systems. Information systems are often accessed and updated from across the globe and utilized by several types of users for many different purposes. In an information systems environment that emphasizes information timeliness, accessibility, accuracy, and maintainability, the data management products of effective IRM policies are vitally important.

2.1.1 Information Resource Management

IRM is a set of policies for the coordinated management of an enterprise's information resources for systems development, operation, and maintenance. IRM policies describe objectives and procedures to provide information availability, timeliness, accuracy, integrity, privacy, security, traceability, ownership, use, and cost-effectiveness [LEFK83]. IRM has been introduced as an organizational unit within many companies and government agencies. For IRM to be effective, it must define a coordinated set of policies that result in integrated systems.

IRM policies provide a structure for coordinated information management, processing, communications, and conversion [NEWM82]. To perform this complex task, IRM combines many disciplines under its extensive umbrella. IRM coordinates the disciplines of Data Administration, Database Administration, Data Processing, Office Automation, Local Area Networking (LAN), Telecommunications, Systems Security, Computer Graphics, Systems Quality Assurance, System Resource Configuration Management, Prototyping, Systems Operations, Project Management, and Systems Integration.

The shift from viewing Data Processing, Data Administration, and Communications as **separate** systems management functions, to viewing IRM as the **synchronized** major policy or function for information management, reflects a change within organizations. Where the traditional Data Processing function tended to emphasize meeting the data **processing requirements** of a single user group, the trend toward IRM policy shows greater emphasis on meeting the many **information requirements** of diverse users.

Data Administration helps IRM fulfill the many information requirements of diverse users. One of the central concepts of IRM, effective information management, is the primary mission of Data Administration.

2.1.2 Data Administration

Data Administration supports IRM objectives by providing data management guidelines and metadata products for effective information modeling, storage, retrieval, security, data validation, and documentation throughout an organization.

Data Administration is the term that is most often used to describe the organizational unit responsible for data element standardization. Other Data Administration functions often include: strategic systems planning, requirements analysis, logical database design, data validation, data security, data change control, data change impact analysis, system resource configuration management, and data coordination among multiple systems and subsystems.

The objectives of Data Administration are to plan, manage, document, and control the data structures underlying the information resources of an organization. The role of the Data Administrator is to integrate and manage an enterprise's information resources, which may be contained in file structures or databases [DURE85]. A Data Administrator may guide the work of several Database Administrators managing separate databases.

A data dictionary system, in its many forms, is the primary tool used by the Data Administrator. The Data Administrator uses a data dictionary system to define and structure metadata (i.e., information about data) for information management. As the data dictionary system standard, the IRDS is designed to support Data Administration. This guide describes how the IRDS supports the Data Administration activities during Strategic Systems Planning.

2.2 IRD Applications for Data Administration

The IRDS supports metadata critical to Data Administration, by providing a medium for metadata description, cross-referencing, and consistency checking. This section describes IRDS support for a number of major Data Administration applications. Later chapters illustrate in greater detail the use of the IRDS in the early development phases of the system life cycle. The following subsections describe some major Data Administration applications that the IRDS can support.

2.2.1 Data Element Standardization

Data elements are the most basic, low-level data structures used in a system. As the result of data element standardization, consistent data elements can be efficiently stored in databases and files, and can be accessed by multiple users and programs. The IRDS supports the standardization of data element names, definitions, and relationships. Data element standardization contributes to the productivity of an enterprise. Figure 3 shows a sample selection from a data element dictionary.

Without data element standardization, different departments and systems of an enterprise can appear to be speaking different languages. A common data element that should be shared by many departments often remains unrecognized because each department calls the data element by a different name. Unless the data element is recognized and standardized, each department must duplicate the effort of collecting, storing, and maintaining the data for that data element. Conversely, misinterpretation problems appear when different departments use the same name for several data elements that have different meanings when used by different departments.

Sample of an IRD
for Data Element Standardization

Entity

Entity = EMP-ACCOUNT-NO
Entity Descriptive Name = EMPLOYEE-ACCOUNT-NUMBER
Entity-Type = ELEMENT
Description = Employee's account number with a financial institution, such as a number identifying a checking account at a bank.

Attributes

Added-By = LAW
Data-Element-Number = 256
Data-Length = 12
Data-Type = NUMERIC
Last-Modified-By = QUINN
Number-of-Times-Modified = 1
Source = PAYROLL-DEPT

Attribute-Groups

Date-Time-Added
- System-Date = 19871020
- System-Time = 092515

Date-Time-Last-Modified
- System-Date = 19871103
- System-Time = 141021

Identification-Name
- Alternate-Name = EMP-ACCT-NO
- Alternate-Name-Context = PERSONNEL-DEPT

Valid-Value-Range
- Lowest-Valid-Value = 1
- Highest-Valid-Value = 999999999999

Relationships

PAYMENT-TO-FINANCIAL-INST REPORT-CONTAINS-ELEMENT EMP-ACCOUNT-NO

DO-PAYROLL-ACCOUNTS SUBROUTINE-ACCESSES-ELEMENT EMP-ACCOUNT-NO
- FREQUENCY-OF-ACCESS = 2 (times-per-month)

MODIFY-EMP-ACCOUNTS SUBROUTINE-UPDATES-ELEMENT EMP-ACCOUNT-NO
- FREQUENCY-OF-UPDATE = 2 (times-per-month)

DEDUCTN-DEPOSIT-TEMPLATE TABLE-CONTAINS-ELEMENT EMP-ACCOUNT-NO

Figure 3

The IRDS helps the Data Administrator find, analyze, and consolidate data elements so that unintentional **synonyms** (i.e., elements with different names but the same meaning) can be located and eliminated. Through the listing of element access names and descriptive names, users can locate typographical errors and unintentional synonyms so that these elements can be consolidated. The IRDS protects the user against homonyms (i.e., elements with the same name but different meanings) which the IRDS will not accept. The resulting standardized data elements become the building blocks for the various databases within an enterprise's information systems.

The Data Administrator can use the IRDS to capture a wide variety of **data element information**, such as: data element access name, descriptive name, alternate names, data element definition, data type, data length, storage format, data validation rules, source (e.g., department that generates the data element), location (e.g., databases or files where the data elements are stored), user access and modification permissions, programs (e.g., programs that use the data element), data element security levels, relationships among data elements, etc. Data element information can be captured from either **top-down** or **bottom-up** system design techniques, or from **reverse engineering** techniques used to document the structure of an existing system.

2.2.2 Database Validation

Data integrity rules to ensure database validation can be represented in an IRD. For a particular entity-type, the user can define data integrity rules such as a **range of values** or a particular **set of values** that are permissible for any instance of such an entity in a database. Once the Services Interface of the IRDS is complete, the user's DBMS will be able to refer to an IRD for permissible data values when data is presented for entry into a database. By comparing the candidate data entry with the data integrity rules stored in the IRD, the DBMS will be able to reject data entries that do not conform to the data integrity rules. Through user specification of data integrity rules in an IRD, the IRDS will assist DBMSs in enforcing data validation.

2.2.3 System Planning Information Management

The IRDS provides a repository and source for system planning and requirements information that can be maintained relatively easily to remain always up-to-date. The IRDS can be instrumental in structuring an enterprise's Business Model and Information Architecture during Strategic Systems Planning. Planning and requirements information stored in an IRD can be readily accessed and updated by many analysts, programmers, and contractors, who can use one IRD at the same time.

The IRDS supports system analysts in defining and cross-referencing diverse and complex functional and data requirements. This cross-referencing results in easier function and data integration, and in better definition of strategic systems plans and system requirements.

Strategic Systems Planning for an entire enterprise is difficult to perform manually because of the contradictions inherent in trying to coordinate and standardize organizational methods, terms, and systems. Prior to Strategic Systems Planning, the organizational units within the enterprise often use different terms and methods for performing their separate and overlapping functions. If the enterprise is large, a high volume of planning information can be encountered. Faced with such a host of organizational discrepancies to be resolved, the planner needs the assistance of a tool like the IRDS.

Information system **Requirements Definition** is difficult to perform manually because of the enormous amount of information to be handled and the complexity of the system to be defined. The IRDS gives the analyst an effective means of describing, comparing, and interrelating all system requirements.

Without the support of a data dictionary system like the IRDS, analysts usually record system requirements on paper, in files, or in databases. System requirements documents produced in this manner are notoriously huge, incomplete, inconsistent, and difficult to understand. In addition, many requirements documents are outdated almost as soon as they are issued, as additions and modifications continue to be defined.

Storing requirements information in files or databases may allow data access and updating, but files or databases do not support many needs for requirements analysis. They do **not** help system designers **identify inconsistencies** that occur between functional and data requirements. Files or databases do not readily represent **interrelationships among requirements.** They cannot effectively relate program and logical database designs backward to requirements nor forward to implementation.

Analysts need the cross-referencing capabilities of a data dictionary system, like the IRDS, to be able to understand how a planned system's functional and data requirements interrelate. IRDS cross-referencing capabilities are based on the structure of **relationships between entities.** For cross-referencing purposes, these relationships can exist between entities in the **same life cycle phase partition** of an IRD, and between entities in **different life cycle phase partitions** of an IRD. With the IRDS Life Cycle Facility that supports cross-referencing of entities across phases, analysts can trace the progress of individual planning elements and system requirements through the later system specification, design, and implementation phases.

2.2.4 System Performance Analysis

The IRDS can support performance information for the subsystem components of an information system. An IRD application can be constructed to capture workload and response time information for database, communications, and data processing subsystems. System designers, programmers, and engineers can use such an IRD to isolate bottlenecks and optimize system performance.

At present, performance information must be entered into an IRD by the user through either the Command Language Interface or the Panel Interface, or by a program through the Application Program Interface. Although performance information collection can be automated, at this time the standard does not specify a facility to enable the IRDS to interface with DBMS services such as the Structured Query Language (SQL).

The Services Interface is being developed for addition to the IRDS standard. When it is adopted as part of the standard, the Services Interface will enable users to automate the capture and access of DBMS performance information to be stored in an IRD. At that time, this IRD can be accessed by the DBMS in active mode.

2.2.5 Data and Function Analysis

The IRDS provides documentation and cross-referencing support for any system development life cycle phases that the user defines. With the IRDS Life Cycle Facility, the system designers can create phase-related partitions within an IRD. In these partitions, system designers can store the information appropriate to each life cycle phase, yet still be able to show cross-reference traceability across phases.

Due to its extensible schema capability, the IRDS can support a variety of information representations, including structures for: functional definition, functional decomposition, data flow analysis, data decomposition, requirements definition, data element standardization, and cross-referencing both within and across phases.

Without the cross-referencing traceability that the IRDS provides, system designers are faced with the arduous task of manual data and function integration. Manual data and function integration for large systems is so difficult that it is rarely performed well. System implementation suffers as a result. Use of the IRDS can improve the quality and reduce the difficulty of data and function integration. Improved data and function integration results in improved system design and implementation.

2.2.6 System Resource Configuration Management

Organizations have long understood the necessity of managing changes to their business, personnel, and monetary assets. More recently, enterprises have begun to recognize the **need to manage changes within their information system resources.** System Resource Configuration Management includes the management of hardware, software, and data resources.

Configuration Management of **other** types of resources involves tracking changes in the number and description of various configuration items, such as the component parts of ships or automobiles. System Resource Configuration Management, similarly, involves tracking structures and changes for all the resources used in one or more information systems.

Configuration management of information system resources is cost-effective when an organization: (1) has one or more large information systems; (2) uses a number of information or graphics storage methods; (3) frequently adds new resources; or (4) frequently modifies or performs maintenance procedures on resources of existing information systems. The IRDS can assist in planning, monitoring, and controlling these resources.

Information system resources include: data resources (such as databases and files), graphics storage media, computer system software (such as computer languages, DBMSs, and software applications), computer systems hardware, system peripherals, and communications systems software and hardware.

Metadata collected about a **configuration item** could include descriptions of: configuration item type, item identification code, item description, primary use, manufacturer, storage format, storage location, security level, acquisition date, operator's documentation, system documentation, maintenance schedule, date of last maintenance, last maintenance type, use restrictions, and responsible department or organization.

An IRD can be used as both a **directory** to **locate** data or item sources, and as a **dictionary** to **describe** configuration item information. The directory and dictionary functions can be combined in one IRD, or can be divided among two or more IRDs.

2.2.6.1 Data Resource Management

An IRD used as a **Data Resource Directory** describes the **use, location,** and **mode of data representation** of on-line data files, on-line databases, paper-based engineering drawings, computer graphics images, archived data on tape or disk, paper records, database reports, spreadsheet generated reports, word processing files, etc.

2.2.6.2 Software Resource Management

Software Configuration Management describes the use, location, and structure of reusable software items. To support this task, an IRD can be used as a **Source Code Directory** that lists and locates the programs, subroutines, functions, software utilities, and languages available in one or more **software libraries.** Software system operator's documentation can be represented in an IRD **Software Operations Dictionary** to describe how to use the information system software items listed in the source code directory.

An IRD can serve as a **Source Code Documentation Dictionary.** The software documentation represented in such an IRD would allow information systems designers and maintainers to see and analyze the **structure** of the system's programs **without having to read** the **documentation embedded in the code.** A source code documentation dictionary would contain the software item's **short name**, its full **name**, its **purpose** (e.g., data conversion, screen windowing, programming language translation, etc.), its **type** (e.g., main program, function, subroutine), the program the software item is **called by**, the subroutines or functions the software item **calls**, the **parameters** passed into the software item, a **description** of the processes performed by the software item, and the **variables** that the software item returns.

2.2.6.3 Hardware Resource Management

Hardware Configuration Management describes the use, location, and structure of computer and communications hardware used to support an information system. An IRD can be designed as a **hardware directory** to list and locate hardware, peripherals, testing equipment, and backup items required for computer and communications systems maintenance, replacement, and repair. Hardware system operator's documentation can be represented in a **Hardware Operations Dictionary** that describes how to use the hardware items listed in the hardware directory.

To manage system hardware configurations, an IRD tracks all computer and communications hardware items acquired, the location of each hardware item, who has responsibility for each hardware item, how each hardware item is used, and all upgrades, scheduled maintenance, and repairs to the hardware item.

An IRD can serve as a **Systems Architecture Documentation Dictionary.** The hardware documentation represented in such an IRD would allow information systems designers and maintainers to understand the **structure** of a system's hardware architecture, without having to go to a written report. A hardware maintenance documentation dictionary would contain the hardware item's **identification**, its **manufacturer and model number**, its **purpose**

(e.g., number crunching, file management), its **type** (e.g., multiplexer, front end, mainframe), the hardware item's **connection mode** to other hardware items, the **purpose of the connection**, and a description of the **major software systems** that the hardware item supports.

An IRD can be modeled to represent a **Hardware Maintenance Dictionary** containing descriptions of the **maintenance schedule** and **maintenance procedures** for each hardware item, a log of **hardware failures**, and a description of the **unscheduled repair, upgrade**, and **maintenance procedures** performed.

2.2.7 Distributed Database Directory

Distributed database are buzzwords of the 1980s and 1990s. Since many organizations have multiple databases, files, and data processing sites, users have problems in finding the correct place to search for particular data.

Distributed data management has become necessary to help users find data among many database locations, and to keep data consistent among many locations. The concept of distributed data management is to provide automated data access and updating among multiple databases and files during system operation.

To locate data entities stored in different databases, the emerging **distributed DBMS software relies on an active, embedded data dictionary system.** This data dictionary system provides a directory of data entity locations, so that the DBMS can easily find each data entity in the correct database.

At present, a directory of data entity locations can be represented in an IRD, along with a description of each entity's mode of representation in a database, file, or other form. When the Services Interface is incorporated into the IRDS, the IRDS can then be used in an active mode with a DBMS to support distributed database management.

An active IRD will also be able to support distributed data management for file management systems. Other additions to the IRDS standard are being considered to further support distributed database management.

2.3 Evolutionary Stages from Data Processing to IRM

The adoption of the IRDS standard is a step forward in the evolution from single system Data Processing to integrated systems Information Resource Management (IRM). The IRDS standard provides a foundation for the widespread commercial development and use of high-quality, compatible dictionary systems.

IRDS standardization can be viewed as a step toward the data processing systems "maturity" depicted in Richard L. Nolan's Stage Theory. Figure 4 shows a modified version of the Nolan Stage Theory Model [NOLA82], [NOLA79], [FONG86]. The model has been modified for this guide. A column for "Data Dictionary Systems" has been added, and the last row has been renamed, changed from "Maturity" in the Nolan model, to "Information Resource Management" here. Other terms have also been modified.

The six stages of the model in Figure 4 indicate an organization's progression in Data Processing stages from infancy (i.e., Initiation) to maturity (i.e., Information Resource Management). The six column titles provide a framework to interpret the growth from stage to stage.

2.3.1 Systems Initiation

In stage one, computer hardware and software are just being acquired or developed. Software applications are designed to solve simple calculation and sorting problems, and to reduce costs for the organization. Since the programs are few and are relatively limited in scope, systems planning and control is not yet a necessity.

2.3.2 Systems Contagion

By stage two, the news of Data Processing success has spread, resulting in a flurry of intense system development. More hardware and software is acquired, and software applications proliferate. Every department wants one or more systems of their own, for prestige as well as for cost savings. A backlog of requests begins to mount in the Data Processing department. As more systems are developed, discrepancies in data naming and usage appear between one system and another. The Data Processing department may invest in a stand-alone data dictionary system in an attempt to reduce the confusion among overlapping systems.

2.3.3 Systems Control

In stage three, the Data Processing department begins to realize the need to control the system development process. Instead of developing a separate system for each user, perhaps some users could share the same system. Some parts of some systems might be interchangeable, so that a programmer might be able to add onto an existing system to satisfy the requirements of another user. The Data Processing department is more likely to use one or more stand-alone data dictionary systems, and may begin to maintain data element dictionaries for larger systems.

Organizational Stages toward IRM

Stage	Purpose	Applications	DP Planning & Control	Control System Objectives	Data Dictionary Systems
1. Initiation	Acquire Computer Hardware & Software	Functional Cost Reduction Applications	Lax	None	None
2. Contagion	Develop Many Data Processing Systems	Proliferation of Applications	More Lax	Facilitate Growth	Emerging Stand-Alone Data Dictionary Systems
3. Control	Develop Controls to Systems Growth	Upgrade Documentation & Restructure Existing Applications	Formalize Planning & Control	Improve Quality & Contain Systems Supply	Increased use of Data Dictionary Systems
4. Systems Integration	Begin Systems Integration & Data Sharing	Database	System Tailor Planning & Control	Improve Performance, Match Supply & Demand	Data Dictionary Systems Integrated with DBMS
5. Data Administration	Promote Data Management & Improved Design	Integrate Application Programs & Databases	Increase Data & Applications Sharing	Improve Quality & Performance, Contain Demand	Data Element Standardization & Other DA Applications
6. (Maturity) Information Resource Management	Design, Develop, Operate & Maintain High Quality Information Systems	Information Systems Include Data Management, Processing, Conversion & Communications	Strategic Systems Planning, Resource Management, Function & Process Integration	Support System Integration with Planning, IRDS, Conventions & Methodologies	IRDS Support for Systems Planning, Design, Development, Operations, & Maintenance

Figure 4

2.3.4 Systems Integration

In stage four, the need to integrate systems becomes apparent. DBMS software is acquired, and many databases are developed. Data dictionary systems integrated with DBMSs are acquired. Due to innovative metadata management, several systems from a department now share data from one or more databases. Planning and control is tailored to each system, as needed. As the Data Processing department initiates or adopts these metadata management controls, it begins to match system development to user demand.

2.3.5 Data Administration

By stage five, the organization recognizes that data management and control is the key to greater productivity and improved systems design. From the partial integration of systems in the previous stage, the organization decides that it wants more fully integrated systems. To encourage data consistency and integrity, Data Administration is introduced as a new department within the organization. Several types of data dictionary systems are acquired from the many commercially available.

During this stage, the Data Administration department uses a data dictionary system to maintain Data Element Standardization for all large systems, and possibly for all corporate systems. Many data dictionaries are developed to support different types of metadata. The Data Administration department actively promotes and supports data sharing, data and function integration, and logical database design. Logical database design information and other metadata are supported by data dictionary systems.

2.3.6 Information Resource Management

At stage six, we arrive at IRM. This stage may correspond to the system "maturity" that Nolan described. IRM represents our current understanding of how best to ensure system efficiency and effectiveness. Nolan's final stage of organizational data processing "maturity" may remain elusive, however, with IRM to be succeeded later by enthusiasms for other advancements such as knowledge based systems.

An organization using IRM recognizes the need for Strategic Systems Planning in stage six, when organization-wide, top-down planning for data and system resources is viewed as a necessity. An organization involved in Strategic Systems Planning wants to be able to trace this plan down through the individual features of system requirements, design, and implementation.

Such an organization using IRM recognizes the importance of Strategic Systems Planning: (1) to reconcile the inconsistencies among separate systems and within individual systems, (2) to trace the relationships of data to functions, and (3) to provide continuing management support for information systems that help the enterprise meet defined management objectives. The IRDS standard provides support for these resource planning functions.

Organizations approaching the IRM stage recognize the need for the **integration** of **data management**, **data processing**, **data conversion**, and **data communications** systems into fully functional information systems. Use of the standard IRDS supports information systems integration of these four functions.

3.0 Features of the IRDS Standard

The IRDS supports information management throughout the phases of systems development, as well as during system testing, operations and maintenance. The standard IRDS specifies a basic set of facilities to be supported in the Core IRDS, and a number of more advanced facilities to be supported in optional IRDS modules. Further description of the IRDS Core and modules can be found in [GOLD88a]. Comprehensive IRD applications can be developed through the use of IRDS facilities.

Figure 5 shows how the contents of the IRDS, including the layers of IRD schema description, IRD schema, and IRD metadata, relate to data in production databases. The highest level of the figure, the IRD Schema Description Layer, was not provided in early data dictionary systems.

3.1 History of Data Dictionary Systems

Vendors of early data dictionary systems defined **fixed metadata types** used in those systems, providing a limited number of entity-types, relationship-types, and attribute-types to users, similar to those shown in the IRD Schema Layer of Figure 5. The schemas of these early data dictionary systems were vendor-defined and could not be modified by users. Since vendors could not reasonably anticipate the best metadata types for every user, this fixing of dictionary terminology severely restricted applications developed with such dictionary systems.

In response to user demand, many vendors then added special facilities for **user-defined metadata types**, so that users could create new entity-types, relationship-types, and attribute-types. These facilities for schema modification provided for schema extensibility, however the user-defined metadata types were **poorly integrated** with the built-in metadata types fixed in those systems. Users of such early dictionary systems were required to learn **two different sets of commands**: one for fixed dictionary terms, and another for user-defined dictionary terms.

3.2 Existing Features of the IRDS Standard

The IRDS is the next step in data dictionary system evolution. The IRDS completely **integrates both built-in and user-defined schema facilities.** The user has complete freedom to tailor the IRD to a particular task, while retaining the ability to use all IRDS functionality.

3.2.2.3

The metadata types built-in to the IRDS are incorporated in the Minimal and the Basic Functional Schemas, described below in the section on Predefined Schema Structures. The features of the existing IRDS Standard are discussed briefly in the following sections.

3.2.1 Entity-Relationship-Attribute Modeling

A **schema** is the part of a dictionary or database that defines the basic structures to be supported in that dictionary or database. This section describes the structure of an **IRD schema**, which defines the structures that support metadata representation in an IRD. IRD schema descriptors provide the foundation on which metadata structures can be built. The IRD Schema Description Layer is illustrated as the top level of Figure 5. IRD schema description is based on the Entity-Relationship information model, which was first proposed by Peter Chen [CHEN79] and continues to be developed [TEOR86]. The Entity-Relationship model supports the analysis of entities, relationships, and attributes.

Entity names correspond to nouns. Entities are the **data concepts** that are represented and described in a data dictionary or IRD. An entity can represent people, places, things, organizations, concepts, or events about which an organization collects data. Examples of entities, EMP-NO and SOC-SEC-NO, are shown in the IRD Metadata Layer of Figure 5. The entity SOC-SEC-NO can be captured as metadata in an IRD along with a description such as, "9 digit number issued by the U.S. Social Security Administration to identify U.S. citizens and resident non-citizens who have Social Security accounts." Since the entity SOC-SEC-NO is viewed in terms of its **characteristics** and **mode of representation**, it is considered **metadata**.

Any particular **instance of an entity**, however, is considered to be **data**, or a **data item**, not metadata. For example, the numbers 229-21-5941, 228-70-9850, and 240-39-7633 are all considered data items or **instances** of the entity SOC-SEC-NO. **Data items**, such as these instances of SOC-SEC-NO, should be represented as production **data** in a **database**. Such data items are often used as **keys** to locate an individual's record in a database.

In Figure 5 at the lowest level, Social Security Number 229-21-5941 is shown as production data that is an instance of SOC-SEC-NO. As described in Chapter 1, data items or instances, such as 229-21-5941, belong in a database, not in a data dictionary or IRD which are designed to support metadata.

Information Resource Dictionary System (IRDS)
Contents in Relation to Data

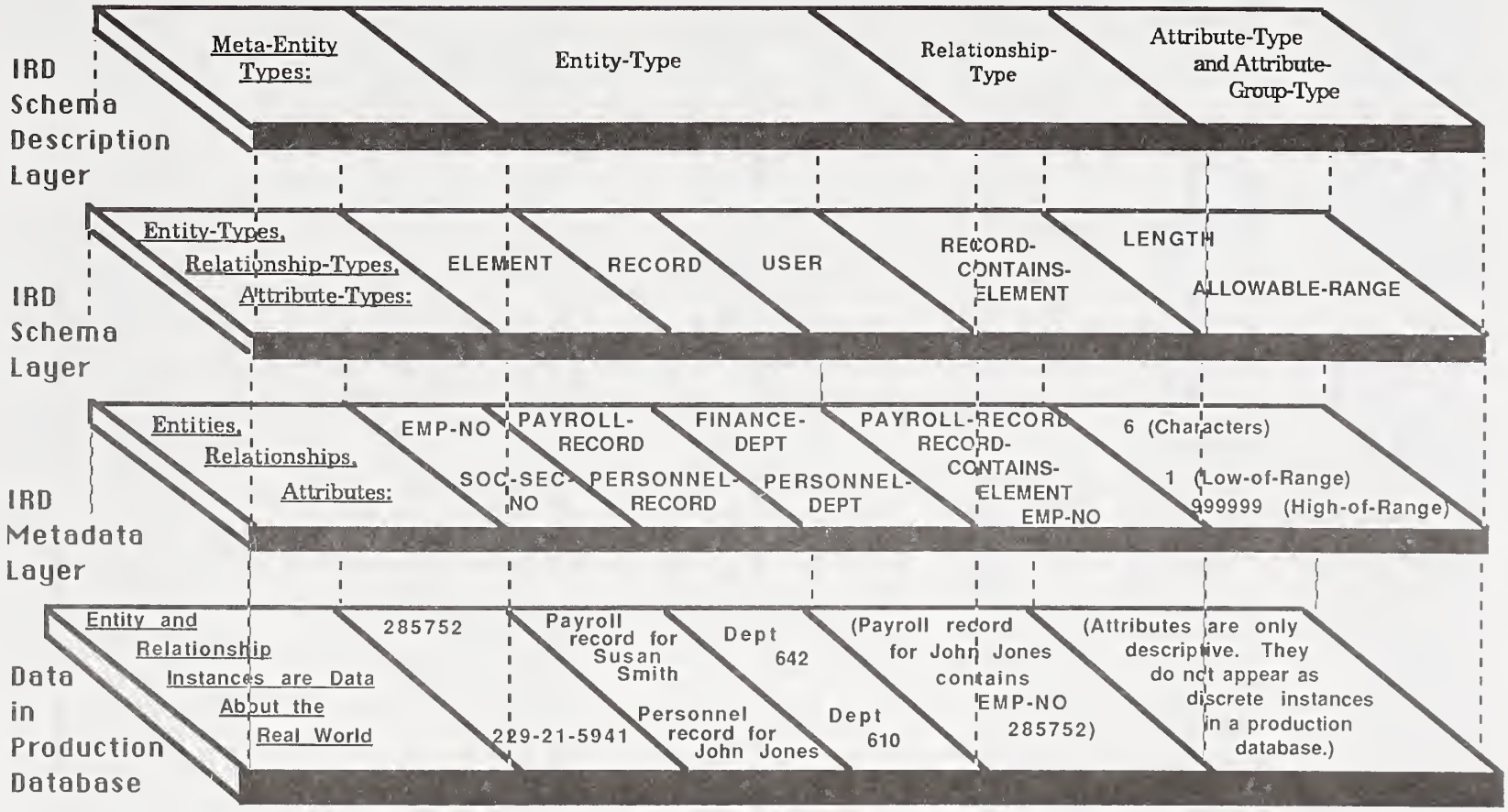


Figure 5

Before an **entity**, such as **SOC-SEC-NO**, can be added to the IRD, however, its **entity-type** must be defined in the dictionary schema. One entity-type is used to establish a category for a number of entities. For example, the entity-type **ELEMENT**, which appears in the IRD Schema Layer of Figure 5, can provide the basis for defining entities such as SOC-SEC-NO and EMP-NO.

Attribute names correspond to adjectives or adverbs, and are used to **describe characteristics** of entities or relationships. As illustrated in the IRD Metadata Layer of Figure 5, the attribute value "9" (characters) of the attribute-type LENGTH describes the entity SOC-SEC-NO.

Attribute-types establish the categories of attributes that can be supported by an IRD. Attribute-types must be defined in the IRD schema before individual attribute values can be added to the dictionary as metadata. As shown in the IRD Schema Layer of Figure 5, an example of an attribute-type is LENGTH.

Relationship names correspond to verbs, and are used to show **how one entity corresponds or associates with another entity.** An example of a relationship is shown in the IRD Metadata Layer of Figure 5, as PAYROLL RECORD-CONTAINS-ELEMENT EMP-NO, where PAYROLL is an entity of type RECORD, and EMP-NO is an entity of type ELEMENT. Before individual relationships can be described as metadata in the dictionary, however, relationship-types must be defined in the IRD schema. Relationship-class-types, which classify relationship-types, may also be defined in the IRD schema.

The **relationship-class-type** option permits the user to define the central relationship term for relationships to be established between entities in an IRD. Examples of relationship-class-type are HAS and CONTAINS. A **relationship-type** establishes the category of association between two entity-types. A relationship-type includes a definition of the first entity-type, the relationship-class-type between the entities, and the second entity-type. As shown in the IRD Schema Layer of Figure 5, an example of relationship-type is RECORD-CONTAINS-ELEMENT.

3.2.2 Predefined Schema Structures

The IRDS provides a number of predefined schema structures. The **Minimal Schema**, included in each IRDS, contains a set of schema descriptors that provide critical schema structures used in every IRD. The **Basic Functional Schema**, an optional module, supplies **commonly used schema structures** that may be applicable to many IRD applications.

The **IRDS Core** supplies the framework for all other predefined and user-defined schema structures. Among the meta-entities defined in the IRDS Core are a number of meta-attribute-types that can be used for IRD **metadata control and partial validation.** A few Minimal Schema structures can be used to enforce **metadata validation.**

3.2.2.1 The Minimal Schema

The IRDS standard specifies a Minimal Schema for every IRDS implementation. The **Minimal Schema**, part of the IRDS Core, provides the critical schema descriptors needed to control every IRD schema and IRD, such as:

- **Entity-types**: IRDS-USER, IRD-VIEW, and IRD-SCHEMA-VIEW
- **Attribute-types**: ADDED-BY, IRD-PARTITION-NAME, LAST-MODIFIED-BY, IRD-SCHEMA-PHASE-NAME, SYSTEM-DATE, SYSTEM-TIME, and VALUE-VALIDATION
- **Attribute-group-type**: DATE-TIME-ADDED
- **Relationship-class-type**: HAS
- **Relationship-types**: IRDS-USER-HAS-IRD-VIEW, IRDS-USER-HAS-IRD-SCHEMA-VIEW

3.2.2.2 The Basic Functional Schema

The **Basic Functional Schema**, one of the IRDS modules, provides an initial set of schema structures that the user can use and build upon. The main schema descriptors specified in the Basic Functional Schema are a set of entity-types, relationship-types, attribute-types, and attribute-group-types, which are described below. For a full definition of the Minimal and Basic Functional Schemas, see [GOLD88a].

The **entity-types** that are predefined for the user in the Basic Functional Schema include: USER, SYSTEM, PROGRAM, MODULE, FILE, DOCUMENT, RECORD, and ELEMENT.

For the entity-types named above, many **attribute-types** are predefined for the user by the Basic Functional Schema, such as: ADDED-BY, ALLOWABLE-RANGE, CLASSIFICATION, DATE-TIME-ADDED, DESCRIPTION, EXTERNAL-SECURITY, LENGTH, LOCATION, and USAGE.

The Basic Functional Schema predefines a number of **relationship-class-types** for the user, including: CONTAINS, PROCESSES, RESPONSIBLE-FOR, RUNS, GOES-TO, DERIVED-FROM, CALLS.

Specific relationship-types between entity-types are also predefined for the user, such as: SYSTEM-CONTAINS-PROGRAM, DOCUMENT-CONTAINS-ELEMENT, PROGRAM-PROCESSES-DOCUMENT, MODULE-PROCESSES-ELEMENT, USER-RESPONSIBLE-FOR-DOCUMENT, USER-RUNS-PROGRAM, PROGRAM-GOES-TO-PROGRAM, etc. The directionality of the required relationship-types is also predefined.

A few attribute-types for relationship-types are defined, such as: FREQUENCY, which is associated with the relationship-types of PROCESSES and RUNS.

3.2.2.3 Schema Structures for Metadata Control and Validation

The IRDS Core provides the foundation for all IRDS schema structures. The IRDS Core also provides a number of predefined meta-entity structures of **meta-attribute-type** designed for **validation and control** of the metadata that will be added to an IRD. Since these meta-entity structures are used to define the schema, **they cannot be changed.**

The IRD Administrator for your organization or your own system, however, **can assign appropriate values** to these meta-attribute-types. For instance, the meta-attribute-type **PURPOSE** can be assigned a textual value that **describes the use** of a particular attribute-type. Meta-attribute-type values such as this provide the basis for describing or controlling the metadata that can be added to the IRD.

While the **value** for the meta-attribute-type PURPOSE is **user-assigned**, the values for some other meta-attribute-types, such as **LAST-MODIFIED-BY**, are **system-assigned.** For LAST-MODIFIED-BY, the IRDS will assign as a value the name of the person who last modified a particular IRD schema or metadata value.

A number of these meta-attribute-types can be used **to store and enforce data integrity rules for the partial validation of IRD metadata.** For example, the meta-attribute-type **FORMAT** can be designated by the IRD Administrator to indicate the **acceptable category of attribute values for a particular attribute-type**, as either: **STRING, TEXT, INTEGER, REAL, DATE,** or **TIME.**

The default value for FORMAT is **STRING**, which permits an attribute value to be a certain type of alphanumeric field. If the FORMAT for an attribute-type is designated as **INTEGER**, attribute metadata can **only** be added in INTEGER form. With FORMAT specified as INTEGER, for example, the IRDS will **protect** the IRD by keeping users from mistakenly adding real numbers or alphanumeric attribute values. By permitting IRD users to add only those attribute values that conform to the FORMAT specified, FORMAT restrictions assist in enforcing IRD data integrity rules.

In addition to these Core structures, a few Minimal Schema structures have been predefined to support **metadata** validation **within an IRD.** The Minimal Schema contains the following meta-entities that can be used to enforce metadata integrity and consistency:

- o **ATTRIBUTE-TYPE-VALIDATION-DATA** - One meta-entity of Attribute-Type-Validation-Data has been predefined as **YES-OR-NO-VALUE** and can be used to validate any attribute value as either **YES** or **NO.**

- o **ATTRIBUTE-TYPE-VALIDATION-PROCEDURE** - Two meta-entities of Attribute-Type-Validation-Procedure have been predefined as:

  - **RANGE-VALIDATION**, which can be used to restrict the attributes of a given attribute-type to a user-defined **range** of values.

  - **VALUE-VALIDATION**, which can be used to restrict the attributes of a given attribute-type to a user-defined **set** of values.

3.2.3 Command and Panel Interfaces

A conforming implementation of the IRDS standard must contain either the Command Language Interface, or the Panel Interface, or both interfaces.

The **Command Language Interface** supports the user's interaction with the IRDS in both batch and interactive modes. The Command Language is fully described in [GOLD88b]. Chapters 4 and 5 of this guide illustrate the use of the Command Language in some IRD applications.

The **Panel Interface** provides a set of panels (i.e., conceptual screens) through which the user can access and manipulate an IRD. After using panels to define particular actions, a user can "save" any number of those panels to be used in a future session. A user can temporarily save or "mark" panels that will be kept only for the duration of the current panel session.

The IRDS standard specifies the functional characteristics of the Panel Interface, without specifying screen design or layout. Different IRDS implementations will have the same panel structural features, although their screens may not look alike.

Each IRDS panel contains the following:

- State Area--reports which IRD is being accessed, what kind of action is being performed, and what system defaults are in effect
- Data Area--shows the placement of input information, or displays output results
- IRD Schema Area--displays those options available to the user with the current schema, or shows the limitations in effect
- Action Area--shows the other panels of the interface to which the user may currently transfer to continue operations, and supports the COMMIT function to perform user-specified updates or retrievals
- Message Area--displays any IRDS error or warning messages
- Help Area--provides information to the user in response to "Help" requests

Sets of panels, called Panel Trees, are included in IRDS specifications to assist users in performing: IRD metadata maintenance, IRD metadata output, IRD entity-lists, IRD schema maintenance, IRD schema output, and the interchange of schemas and metadata between different IRDs.

3.2.4 Extensible Schema Definition Capability

The IRDS permits the user to make essentially unlimited extensions to any IRD schema. This capability provides the user with the flexibility to design a schema to fit the particular needs of the organization or life cycle phase that the IRDS is being used to support. The user has the freedom to extend the schema in any manner that will be useful. This means that the IRD can be structured to capture any type of metadata that the user can define.

All implementations of the IRDS standard will have this extensible schema definition capability, since it is part of the IRDS Core. This flexibility permits vendors to choose whether or not to implement the Basic Functional Schema as a starter schema set, and permits users to customize their IRD applications.

3.2.5 Extensible Life Cycle Phase Facility

Two types of life cycle phase facilities are specified in the IRDS standard. The Core IRDS has a basic life cycle phase facility that provides the user with the capability to construct partitions in an IRD corresponding to the life cycle phases to be represented. Additional facilities are supported by the IRDS Extensible Life Cycle Phase Module. This optional module specifies integrity rules and customization facilities that give the user comprehensive life cycle support.

The Core IRDS provides three life cycle classes in which the user can represent specific life cycle phases. These three classes are:

- **Uncontrolled**--IRDS users can represent any number of system development life cycle phases through the use of this Uncontrolled life cycle class. For example, some development life cycle phases to be represented in this Uncontrolled class might include: Strategic Systems Planning, Requirements Definition, Functional Specification, Program Design, Logical Database Design, Physical Database Design, and System Implementation. Uncontrolled generally describes "non-operational" life cycle phases.

- **Controlled**--This life cycle class can contain only one Controlled Life Cycle Phase, which corresponds to the activities of systems operations and maintenance.

- **Archived**--This life cycle class can contain only one Archived Life Cycle Phase, which corresponds to the documentation activities associated with historical and audit archives for metadata no longer in use.

Beyond the life cycle classes provided by the IRDS Core, the Extensible Life Cycle Phase Module offers three additional capabilities including Hierarchical Phase Modeling, Relationship Sensitivity Structures, and Life Cycle Integrity Rules, which are described below.

3.2.5.1 Hierarchical Phase Modeling

The module permits users to designate hierarchical relationships among the phases. For example, when a system is under development and the developer wants to demonstrate how system requirements are being fulfilled in the implementation, the

developer may want to designate the Requirements phase (in the Uncontrolled class) as the **top of the hierarchy** under which all specification and design phases (also in the Uncontrolled class) fall. Later, when the system is operational, the Data Administrator may want to revise the hierarchical phase modeling to show the Controlled phase (representing System Operation) at the top with development phases, such as Requirements and Logical Database Design (in Uncontrolled class) organized beneath.

3.2.5.2 Relationship Sensitivity Structure

In planning for the movement of metadata from a lower level life cycle class to a higher level class, the IRD Administrator has the option to categorize certain relationship-types, and therefore their corresponding relationships, as **phase-related.** When a phase-related relationship associates two entities, then the **first entity** in the relationship can be considered **"dependent"** on the **second entity**, while the second entity is considered **"independent"** of the first.

For example, suppose that the relationship-type **PROGRAM-ACCESSES-SUBROUTINE** is defined as phase-related in the System Design (Uncontrolled class) phase. In the relationships based on this type, entities of the type **PROGRAM** are considered **dependent** on entities of the type **SUBROUTINE.** This dependency means that in order **to be considered complete**, entities of type PROGRAM **require the presence** of entities of type SUBROUTINE. The phase-related relationship-type establishes this entity dependency.

The **phase-related dependency extends to the specific relationships of this relationship-type.** For instance, in the relationship **PRODUCE-PAYROLL PROGRAM-ACCESSES-SUBROUTINE PREPARE-CHECKS**, the entity PRODUCE-PAYROLL, of entity-type PROGRAM, has a phase-related dependence on entity PREPARE-CHECKS, of entity-type SUBROUTINE. Such phase-related, relationship dependencies provide a foundation for the IRDS to enforce life cycle integrity rules.

3.2.5.3 Life Cycle Integrity Rules

The Extensible Life Cycle Phase module provides life cycle integrity rules intended to protect the IRD when the IRD Administrator moves metadata from an Uncontrolled phase into the Controlled phase, or from the Controlled phase into the Archived phase. Life cycle integrity rules should be used when an IRD Administrator wants to move metadata already defined in one phase to another phase of a higher life cycle class.

In planning for such an occurrence, the IRD Administrator has the option of defining certain relationship-types as "phase-related" in the source phase. Using the Relationship Sensitivity Structure described above, the IRD Administrator has the flexibility to define which sets of entities, associated by relationships, are essential to the coherency and completeness of the target phase.

These phase-related relationships permit the system to **enforce integrity rules** as metadata is moved from the source phase to the target phase. The integrity rules require that **independent entities** must be transferred from the source phase into the higher level target phase **before** the **dependent** entities of that relationship can be transferred, so that their relationship dependencies are retained.

If the relationship PRODUCE-PAYROLL PROGRAM-ACCESSES-SUBROUTINE PREPARE-CHECKS is phase-related, the first entity PRODUCE-PAYROLL is considered dependent on the second entity PREPARE-CHECKS. When the user moves these entities from an Uncontrolled class life cycle phase to the Controlled phase, the **independent entity** PREPARE-CHECKS must be moved to the target Controlled phase **before** the **dependent entity** PRODUCE-PAYROLL can be moved there. This ensures that no entities of the type PROGRAM can be moved to the target phase without having entities of the type SUBROUTINE already present in the target phase.

Since the description of a program could not be considered complete without information about its component subroutines, it is reasonable to require the description of the related subroutines before an entity can be established for a program that calls subroutines. Due to this integrity rule, when the user refers to an entity of the type PROGRAM in the target phase, the user can be assured of finding in place all the information that was originally defined for all related entities of the type SUBROUTINE.

3.2.6 IRD Versions, Views, and Quality Indicators

The IRDS provides specifications for versions, views, and quality indicators. A **version identifier** for entities and meta-entities is composed of a **variation name** and a **revision number**.

The IRDS versioning facility permits **revision numbers** to be assigned to changing versions of each entity as part of the access name. Every entity is automatically assigned a revision number of "1" when it is first defined. When the user specifies a "new-version" during entity modification, for example, that entity's revision number is incremented. Each entity revision is stored separately in the IRDS as a separate entity.

For instance, if you had originally defined an entity EMP-NO with a description showing it to be a 5-digit integer, the IRDS would, by default, assign it a **revision number** of "1" indicating that it is the first version without revisions. This entity name would then be EMP-NO(1). Later when your company had added more employees, if you changed your EMP-NO description showing it to be a 6-digit integer, then you could assign this version to have the revision number of "2" indicating that it is the first revision, or second version, of EMP-NO. This revised entity name would be EMP-NO(2). Unless you remove EMP-NO(1) from the IRD, both EMP-NO(1) and EMP-NO(2) will exist in your IRD.

This facility also permits the differentiation of entities with **variation names.** You can use variation names, for example, to differentiate between entities of the same name that are used in different ways, such as those used in more than one life cycle phase partition of the same IRD.

If an entity such as PRODUCE-PAYROLL, of entity-type FUNCTION, is defined in the REQUIREMENTS-LIFE-CYCLE-PHASE, you may want to use the **same entity name** entity name again in the PROGRAM-DESIGN-LIFE-CYCLE-PHASE, now with entity-type PROGRAM. Unless you alter the entity name with the addition of **different variation names**, the IRDS will **not** permit the definition of the same entity name again in different phase partitions of one IRD.

In the REQUIREMENTS-LIFE-CYCLE-PHASE, you can represent the entity name as PRODUCE-PAYROLL(R), in which the "R" indicates Requirements; in the PROGRAM-DESIGN-LIFE-CYCLE-PHASE, you can represent the other entity name as PRODUCE-PAYROLL(P), in which the "P" indicates Program Design. Two different entities are now represented with the addition of these variation names. Any number of variation names can be defined. Variation names, which must begin with a letter, can be used for a variety of purposes.

Variation names and **revision numbers** can be used together. For example, if the entity PRODUCE-PAYROLL(R) has been revised once, for instance to modify its entity description, then the variation name and the revision name can be represented together as PRODUCE-PAYROLL(R:2).

The IRDS supports the definition of dictionary **views** to provide logical partitions within the Uncontrolled life cycle phases of an IRD. A view is a defined subset of an IRD that is used to provide a limited perspective of the IRD contents. Different user groups will often use different views to access or develop different aspects of the same dictionary.

For example, when multiple groups are developing system requirements using the Requirements Definition phase of an IRD, each user group will want to access only their particular part of

the Requirements Definition dictionary. The Requirements Definition IRD can contain **many views** that may support Software Requirements, Communications Requirements, Hardware Requirements, Site Requirements, etc. The IRD users working on Communications Requirements, for example, would be able to view their metadata separately from all the other metadata stored in the Requirements Definition dictionary.

Quality indicators are supported by the IRDS for use in standardization, quality assurance, and quality testing IRD applications. Quality indicators have not been predefined as meta-entities in either the Minimal or Basic Functional Schema. To use this facility, organizations must define their own set of quality indicator meta-entities in an IRD schema.

Since some organizations must respond to many types of standards, the quality indicator facility provides these users with a method of indicating which type of standard an entity fulfills. Quality indicators can also be used to designate to what degree an entity satisfies quality assurance tests. No integrity rules for quality indicators have yet been provided.

3.2.7 IRD-IRD Interface

The IRDS standard provides general specifications for an IRD-IRD Interface that supports schema and metadata interchange between separate IRDs, which may be located in one IRDS or separate IRDSs. The IRD-IRD Interface facility permits an organization using standard IRDSs to select and transport part or all of an IRD schema, and part or all of the corresponding IRD data, from one IRD to another IRD.

Since the IRD-IRD Interface is defined in the IRDS Core, every standard IRDS implementation has this capability. The IRD-IRD Interface is important because it protects schema and metadata interchange with integrity constraints, so that there is no compromise to the data integrity of either IRD after the interchange. The IRD-IRD Interface is the subject of further standards work to insure interoperability among IRDS products of different vendors.

To transport the contents of one entire IRD to another IRD, the user works from one IRDS to specify the two dictionaries to be accessed. The IRD-IRD Interface compares the schemas of the two different dictionaries, and identifies incompatibilities in the schemas for the user to resolve. Once any discrepancies are resolved and the schemas are compatible, the IRD-IRD interface supports the transport of one dictionary into the other.

Users may find it easier to undertake the IRD-IRD transport incrementally, rather than to combine two complete IRDs. Since many schema-level discrepancies often exist among different IRDs, and because users can more easily resolve a limited number of discrepancies at a time, users may prefer to transport only part of an IRD at one time.

In this case, a subset of the source dictionary metadata is selected and exported to an "empty" IRD. In this new IRD, the user specifies the name of the old source schema, and the IRDS creates a new schema to support the subset metadata. Then the IRDS interface facility compares the schema of the new source (i.e., subset) IRD, with the schema of the target IRD, and reports any discrepancies. When the user has resolved all schema-level discrepancies, the new source IRD and the target IRD can be merged. The user repeats this process until all desired parts of the two IRDs have been merged.

3.2.8 Security Facilities

The Security Facilities Module supports the restriction of user access permissions to an IRD, to an IRD schema, to an entity-type, to individual commands, and to individual entities. Two levels of IRD access control are provided:

- Global Security--This facility provides restrictive mechanisms to protect both the IRD and IRD schema. These restrictions are designated according to entity-type, meta-entity-type, or partition. For each IRDS-USER, IRD-VIEW, and IRD-SCHEMA-VIEW entity, attributes define the user's level of access. A user can be limited to access only a subset of views. Within each view and schema view, individual user permissions (i.e., to read, add, modify, and delete) are defined for each entity-type and meta-entity-type.

- Entity-Level Security--This facility provides the ability to assign read or write privileges for individual entities. As an extension of the Global Security Facility, Entity-Level Security allows read or write 10 digit number "locks" to be assigned by the system to each entity. The system will issue the correct 10 digit number "key" only to users who have been granted the appropriate permissions to access an entity secured in this fashion. The Entity-Level Security Facility requires the use of the entity-type ACCESS-CONTROLLER and a set of relationship-types based on the SECURED-BY class. The "lock" attribute-types associated with the entity-type ACCESS-CONTROLLER are READ-LOCK and WRITE-LOCK. The "key" attribute-types of the entity-type IRD-VIEW are READ-KEY and WRITE-KEY.

3.2.9 Procedure Facility

One of the IRDS optional modules, the Procedure Facility provides the user with the capability of defining and executing new IRDS procedures, or macros, for IRDS commands. The IRDS Command Language Interface Module must be present for the Procedure Facility to operate.

Many kinds of command procedures can be user-defined with the Procedure Facility. Instead of laboriously typing out every command, for example, with the Procedure Facility a user can store sets of commands that perform frequent tasks. Various statement types are permitted in procedures, such as Assignment Statements (i.e., to assign values to variables), Do Statements (i.e., to group instructions together and execute iteratively), If Statements (i.e., to specify conditions for executing procedures), etc.

3.2.10 Application Program Interface

The Application Program Interface provides an interface between standard programming languages and the command language of the IRDS. The Call feature of a standard language, such as COBOL, PL/1 or FORTRAN, can be used to access the metadata in an IRD. The IRDS standard does not specify particular language bindings. With this module, users can write programs to collect metadata from, and pass metadata to, the IRD. Sets of IRDS commands can be passed through the language call to the IRD, and error conditions from the IRDS can be carried back through programs to the user. All IRDS integrity and security rules continue to be enforced with this interface.

3.3 Planned Features of the IRDS Standard

The IRDS is expected to become a widely implemented standard for data dictionary system products. As the IRDS standard is implemented in commercially available IRDS software, IRDS system users will have the benefit of easier system-to-system dictionary transportability.

Specifications are planned for additional IRDS modules. A facility expected to be defined within the next few years is:

- o <u>Services Interface</u> -- This planned facility will permit the IRDS to be accessed by services such as programming language compilers, database query languages like SQL and the Network Data Language (NDL), report writers, and Open Systems Interconnection (OSI) systems. The Services Interface will allow the IRDS to be used actively in an operational environment.

Another facility under consideration for future addition to the IRDS is:

- **N-ary Relationship Module** -- X3H4 has recognized the need for this future module that would permit IRDS users to specify relationships among more than two entities. When this module is implemented, relationships among multiple entities could be defined. X3H4 is considering the n-ary concept to provide the IRDS with a more convenient capability to represent complex relationships. The intent is for this future module to specify the schema descriptors necessary to define an n-ary schema and support n-ary schema extensibility.

4.0 Standards and Conventions for IRD Use

The IRDS protects the user by allowing only one instance of each entity name and relationship name, so that subsequent additions will not inadvertently override previous entity definitions. The IRDS also provides automated support for some areas of metadata protection, such as Global Security and Entity-Level Security supported by the Security Facilities module discussed in Chapter 3. IRDS support for versioning, views, and quality indicators provides the user with a number of devices to protect metadata stored in an IRD.

The IRDS cannot, however, protect the user from himself. When one user constructs and uses an IRD, use of an IRD appears simple. When many users have access to an IRD without limitations or structured procedures, chaos will reign.

4.1 Why Standards and Conventions are Necessary

Your IRD is a mirror of what you put into it. You can lose metadata from your IRD if your user group does not plan ahead for your IRD application's **purposes** and **results.** Without knowing the purposes and planned results of your IRD, your user group can easily develop incorrect schemas for your IRD, necessitating deletions and additions after metadata has been added to the IRD.

Disorganized output will result from your IRD if your user group has disorganized **metadata input procedures.** If you have not defined **areas of responsibility** and enforced these with **security measures**, or if you have not defined an **analysis methodology** to be used by all user group members in developing the metadata, your IRD metadata will not be dependable. A group using an IRD without **naming conventions** and **metadata validation routines** should expect to have redundant, contradictory metadata.

The IRDS provides the user with many facilities, but facilities alone do not ensure a useful IRD. When an IRD has more than one user, you should establish and enforce standards and conventions for its use [VAND82].

An organization interested in using the IRDS should first assign a **small working group** to use the IRDS to gain familiarity with it. Through guidance, such as the use of this guide, and through trial and error, this working group should devise its own set of standards and conventions as the group develops several IRD applications for demonstration purposes.

If an organization has previously used a number of data dictionary or other design tools for which a set of standards and conventions has already been instituted, these existing standards and conventions should be examined and, as appropriate, modified for use with the IRDS.

When this small working group has devised a number of test applications using one or more IRDs, this group should plan a set of **IRDS usage standards and conventions** for use throughout the organization. The following sections provide general guidance on the contents of such a standards and conventions document.

4.2 Standardize Methodologies

The standardized use of **one analysis methodology** by all user group members working in **one system development phase** is critical to the success of any IRD metadata application. Once you have defined the **purpose** and **results** for your IRD, you know your destination. Without a shared methodology, however, your user group will not have the **means** to get there. A standardized methodology provides a set of analysis procedures that all members of a working group can follow during a system development phase to derive the metadata that the IRD will store.

Many methodologies address single life cycle phases. For example, NBS recommends a methodology for use in the logical database design phase, described in Guide on Logical Database Design, NBS Special Publication 500-122 [FONG85]. While methodology **selection** is important, equally critical is methodology **coordination**. When different methodologies are used for different life cycle phases, methodology use must be coordinated so that no information is lost or duplicated among phases. Organizations should adopt and standardize a set of methodologies that have proved successful, for use throughout all the life cycle phases.

The IRDS can be used with any analysis methodology that your organization finds satisfactory. In order to input your analysis into an IRD, however, the metadata resulting from your analysis methodology must be mapped into the Entity-Relationship model for input to an IRD. Mapping of analysis results into the Entity-Relationship model has been discussed earlier in this chapter.

4.3 Share Standardized Schema Structures

A number of user groups can share one IRD, if their purposes and planned results for using the IRD are related. An IRD can be **partitioned** into a number of **phases** in which metadata can be collected to represent the various life cycle phases. Within

each IRD phase, a number of separate user views can be defined to organize metadata according to the focus of working groups that are each concentrating on one area of a phase.

One IRD can support the metadata generated by each requirements group through the use of views that provide a localized perspective of the IRD metadata. Separate user views of one IRD can provide integrated support for a number of groups working on different analysis aspects of a phase.

When developing an information system, an organization may decide to define a number of views, such as a Strategic Systems Planning Phase, a Requirements Phase, a Functional Specification Phase, a Logical Database Design Phase, and a System Design Phase.

A number of different types of Requirements Phase information for a planned system can each be associated with a view defined within this phase of one IRD, so that several groups can work on different requirements issues at the same time without hindering each other.

Within the Requirements Phase, a Communications View can be designated to support a group working on communications requirements, a Decision Support System View can be defined for a group working with that focus, a System Security View can be defined for a group working with that focus, and a Software Configuration Management View can be defined for a group working with that focus. The metadata stored within different views of one IRD phase can be retrieved either from within a view, or from across different views.

One benefit of using multiple views within an IRD phase, rather than using multiple IRDs, will be the ability to cross-reference information generated by different groups. If a user group has defined its own views and has permission to "read" the other views within the one IRD, that user group can then choose whether to limit its perspective to only its own metadata, to show the relationships between its own metadata and that of another group, or to show relationships to all the metadata in the IRD.

For instance, within the Requirements Phase of an IRD defined with separate views, the communications group can choose to limit its metadata definition and retrieval to Communications View metadata. The group can also define and retrieve metadata to show how requirements added through the Communications View can support requirements added through the Decision Support System View. The manager of the requirements analysis project will be able to query and receive reports on metadata stored in this one IRD when checking all requirements areas for completeness.

Another benefit of using phases and views within one IRD, rather than using separate IRDs, will be the shared use of some schema structures common to a number of user groups. This sharing of some schema structures will require better **coordination** among working groups to avoid inconsistencies and duplication of effort. Establishing **security controls** for the shared metadata is also important, to avoid having a situation in which one group accidentally modifies another group's metadata.

When several user groups have different views of one IRD, a special **planning and coordination function** is needed to decide the use of some shared schema structures, and to monitor the development of relationships between entities of different working groups. This planning and coordination function may involve the services of an **IRD Administrator** (i.e., Data Dictionary Administrator) responsible for one or more IRDs and familiar with the IRDS software, and a **Data Administrator** familiar with the metadata subject matter of the system development effort.

4.4 Ensure Responsibility, Efficiency, and Accuracy

When using an IRD, particular areas of responsibility should be assigned to different users, IRD procedures should be planned for maximum efficiency, and metadata integrity rules should be defined. Assigning **responsibility** ensures that users understand, review, and take responsibility for categories of metadata.

4.4.1 Assign Responsibility

Organizations that plan to use the IRDS extensively will wish to assign an **IRD Administrator**, or IRD Administration team, to provide expertise with the IRDS and have responsibility for schema definition, security permissions, and other functions such as interfacing IRDs. A **Data Administration** team, who may often be the primary users of an IRD, can provide useful support to other IRD users for metadata analysis. The IRD Administrator and Data Administration team could be assigned joint responsibility for writing the IRDS standards and conventions guide for use throughout a project or organization.

Individual users and user groups should also be assigned areas of responsibility appropriate to their use of an IRD. Areas of responsibility can be enforced with the Security Facilities module. A person responsible for a certain area of metadata can be assigned **security permissions** that will give that person control of the accuracy of that metadata. The IRD Administrator and Data Administration team should work together to decide when certain security permissions should be granted or withheld.

When security privileges have been restrictively defined and a user wants to **modify an entity** which **another person controls**, the user has several options. The user who wishes to make the change to the restricted metadata must either: (1) submit the change to the responsible person who can decide whether to make the change; (2) submit the change to the IRD Administrator who can decide whether to make the change; or, (3) request the IRD Administrator to grant the user security privileges to that restricted area of the dictionary. These measures help control the quality of the metadata in an IRD.

4.4.2 Ensure Metadata Integrity

Automated **metadata validation checks** should be used to assist in ensuring the **metadata integrity** of the IRD. The IRDS supports a number of validation rules designed to protect an IRD. In addition to checking that metadata names are unique, the IRDS provides other schema structures that can **enforce user-specified integrity rules** for metadata validation. These schema structures are discussed in Chapter 3 in the section on Schema Structures for Metadata Control and Validation.

In addition to using the automated metadata validation facilities supported by the IRDS, organizations should require frequent **IRD reviews** to monitor IRD content. These reviews will assist in ensuring IRD **accuracy and integrity.** Regular **walk-throughs** of IRD metadata should be conducted by the Data Administration team as another means of ensuring the quality of the IRD. Critical walkthroughs of particular metadata areas should be lead by those users who have been assigned responsibility for that metadata. Such walkthroughs provide management and other project groups with information on the **status** of a particular phase or area of an IRD, and provide IRD users with a **critique** of metadata development and IRD practices.

4.4.3 Standardize Efficient Procedures

For an organization to develop and maintain an IRD with **efficiency**, the organization should **standardize procedures** for schema definition and metadata input, update, deletion, and output. Even when all IRD users know **how** to implement these IRDS commands, standardized procedures will show users **when** these procedures should be performed, and in **what manner** these procedures should be carried out.

For example, an organization may prefer that all IRD inputs and modifications be performed in **batch mode at night.** This procedure would permit the IRD Administrator to review all users' metadata definition and modification commands at the end of the day, and to review the results in the morning.

If some individuals in this organization make their IRD changes in **interactive mode**, however, the Data Administrator will not be able to review changes to the IRD as planned. Since the IRD Administrator would only expect changes made in batch form, the interactive modifications could be overlooked. Unrecognized integrity problems may occur in the IRD metadata as a result.

To enforce protections against metadata integrity problems, the IRD Administrator and Data Administration team must **identify potential problem areas** such as this. Once procedures have been specified for IRD users, protective measures can be carried out to see that specified procedures are followed. In addition to standardized procedures for users, **metadata integrity rules** and review procedures should be specified **within an IRD** to validate metadata input and modifications. Data integrity rules for IRD metadata validation are discussed above and in Chapter 3.

An organization's standardized procedures and protections should be fully described in a **standards and conventions document**. This document should designate areas of responsibility, indicate the assignment of security permissions, define procedures for IRD use, and describe the definition of automated metadata validation checks. This standards and conventions document should be required reading for any IRD user, and its procedures should be strongly enforced by the organization.

4.5 Define Naming Conventions

When more than one person uses an IRD, naming conventions become a necessity. Naming conventions should be specified in an organization's standards and conventions document so that IRDS users can avoid the climb up the Tower of Babel.

Syntax conventions for types of names should be defined according to placement of parts of speech within a name. For example, an organization may wish all **entity names** to have the syntax where the first term is an **adjective** directly followed by a noun. Other organizations may want every entity name to begin with a noun. Some organizations may want all entity **access names** to have one syntax format, with the same or another syntax format used with **descriptive names**. **Alternate names** reflect entity usage in various systems and are not necessarily standardized.

Naming conventions may be organized differently according to entity-type. For example, for entities of entity-type **TABLE** a certain syntax could be specified, along with the caution **not** to use abbreviations. For entities of entity-type **FIELD**, on the other hand, another syntax could be specified, along with the recommendation to use standard abbreviations.

Organizations may want to standardize relationship names by establishing a limited set of **relationship-type** names that can be used in an IRD. While some organizations may require only **active verbs** in relationship-types, other organizations may accept either active verbs or **passive verbs.**

Similarly, organizations can require that standardized sets of **attribute names** and attribute-group names be used with particular entity-types. If **attribute-group-types** have particular significance within an organization, such as RANGE-OF-LENGTH or RANGE-OF-WEIGHT, the organization may want to specify entity-types for which these attribute-group-types must be used.

Conventions for forming **abbreviations** will be useful to IRD user groups attempting to develop brief but meaningful access names. Some organizations may want to use an abbreviation dictionary for reference, with the addition of the organization's own abbreviation rules for terms not listed in the dictionary. Those abbreviation rules may require, for example, the formation of all unlisted abbreviations by deleting all vowels from a word.

Along with abbreviation conventions, standard lists of codes and acronyms should be included in your naming conventions. Acronyms permeate our lives in increasing numbers. Standard lists of **acronyms** should be included in naming conventions so that all IRD users know how to use them correctly.

In the interest of IRD economy, **codes** may often be used in an IRD to represent common classifications of metadata. Codes may be standardized by an organization for use in an individual IRD, or may be standardized across all IRDs. As an organization becomes familiar with recurring patterns in metadata, IRD users may begin to recognize opportunities for greater efficiency through the use of codes. Codes should be identified in the naming conventions.

For further discussion of entity naming conventions, see **Guide to Data Entity Naming Conventions**, NBS Special Publication 500-149 [NEWT87]. For a reference book to abbreviations and acronyms, see [CROW84].

4.6 Standardize Data Elements

Data element standardization is performed to ensure that the data used by one or more information systems can be maintained and accessed accurately, effectively, and efficiently. As discussed in Chapter 2, a data element is the most basic form of data used by programs and operational databases.

Data elements must be standardized to permit multiple programs and databases to refer to data in standard forms with standard meanings. To perform data element standardization, organizations collect information about data, or metadata, in one or more repositories where each data element can be defined, described, and cross-referenced to many other data elements and to metadata of other types. An IRD provides full support for data element standardization.

In an IRD used for data element standardization, data elements are defined according to a format prescribed in the organization's standards and conventions document and supported by the IRD schema. All element names should conform to the organization's standard naming conventions.

While data dictionary formats for data element definition vary [VAND82], most data element IRDs should contain information similar to the descriptive format described below. An organization should define dictionary terminology for metadata types that will be most appropriate in its data element standardization applications. An IRD application used for data element standardization might include the following data element format:

Data Element
Entity Information

- **Entity** -- the access name of the data element; a data element is always an entity; entity-type is predefined.
- **Entity Descriptive Name** -- the formal name of the data element entity; this capability is predefined.
- **Entity-Type** -- for data elements, you can use the predefined entity-type ELEMENT or can define another entity-type DATA-ELEMENT.

Data Element
Attribute Information

- **Added-By** -- a predefined, system-maintained attribute-type that identifies the person who first defined the data element; corresponds to another common attribute term that you can define, Responsible-Person.
- **Data-Element-Number** -- you can define this attribute-type to assign an identification number to a data element for additional reference.

- **Data-Length** -- you can define this attribute-type to capture information about the number of characters or digits permitted in data values of the element; a similar attribute-type, LENGTH, has been predefined.

- **Data-Type** -- a predefined attribute-type used to capture information on whether the permitted data values for the element are alphabetic, numeric, alphanumeric, or in date format. Data-Class is a similar predefined attribute-type that you may prefer to use.

- **Description** -- a predefined attribute-type that supports a narrative description of the purposes and uses of a data element entity.

- **Last-Modified-By** -- a predefined, system-maintained attribute-type that identifies the person who last updated the data element.

- **Number-of-Times-Modified** -- a predefined, system-maintained attribute-type that records the number of times an entity, such as a data element, has been modified.

- **Source** -- you can define this attribute-type to identify the organizational unit, such as department, that first defined the data element.

- **Valid-Value** -- you can define a multiple attribute-type like this one to record a set of permitted data values that are either numeric and non-contiguous, or non-numeric; if the permitted data values are numeric and contiguous, use Valid-Value-Range. Allowable-Value is a predefined attribute-type that you may prefer to use.

Data Element Attribute-Group Information

- **Date-Time-Added** -- a predefined attribute-group-type that identifies the date and time that an entity was added; it contains two attribute-types, System-Date of the entity addition, and System-Time of the entity addition.

  - **System-Time** -- a predefined attribute that represents the time that the entity was added, in hours, minutes, and seconds, in the numeric form of HHMMSS.

- System-Date -- a predefined attribute that represents the date that the entity was added, in year, month, and day, in the numeric form of YYYYMMDD.

o Date-Time-Last-Modified -- a predefined attribute-group-type that identifies the date and time that an entity was last modified; it contains two attribute-types, System-Date (of entity modification) and System-Time (of entity modification).

  - System-Time -- a predefined attribute that represents the time that the entity was last modified, in hours, minutes, and seconds, in the numeric form of HHMMSS.

  - System-Date -- a predefined attribute that represents the date that the entity was last modified, in year, month, and day, in the numeric form of YYYYMMDD.

o Identification-Name -- a predefined attribute-group-type that supports alternate names used to identify entities; it contains two multiple attribute-types, Alternate-Name and Alternate-Name-Context.

  - Alternate-Name -- a predefined multiple attribute-type that records aliases that can be used, instead of an entity's access name, to specify that entity; any number of Alternate-Names may be defined; each Alternate-Name is paired with an Alternate-Name-Context.

  - Alternate-Name-Context -- a predefined multiple attribute-type that records the context in which an Alternate-Name is used, such as a department or a system; one Alternate-Name-Context corresponds to one and only one Alternate-Name.

o Valid-Value-Range -- you can define an attribute-group-type like this one to record the range of valid data values permitted for a data element; this definition of valid data values provides useful information for data validation; valid-value-range can be used if the permitted data values are a contiguous, numeric set; if the permitted data values are non-numeric, or numeric but non-contiguous, use a multiple attribute-type such as Valid-Value. Allowable-Range is a predefined attribute-group type that you may prefer.

- **Lowest-Valid-Value** -- you can define this attribute-type to record the lowest permitted numeric value in the valid range. Low-of-Range is a predefined attribute-type that you may use with Allowable-Range.

- **Highest-Valid-Value** -- you can define this attribute-type to record the highest permitted numeric value in the valid range. High-of-Range is a predefined attribute-type that you may use with Allowable-Range.

Data Element Relationship Information

o **Process-Accesses-Element** -- you can define this relationship-type; for a system being developed, this relationship-type permits the user to identify the processes that access this element without updating it; the description of a process is captured in a separate entity-type, Process; the location of a process, in turn, can be captured in relationships such as System-Contains-Process and Department-Performs-Process; if the system being described is operational, use the relationship-type Subroutine-Accesses-Element instead.

  - **Frequency-of-Access** -- you can define this relationship attribute-type to describe the frequency with which the process is expected to access the element.

o **Process-Updates-Element** -- you can define this relationship-type; for a system being developed, this relationship-type permits the user to identify the processes that update a particular element; as explained above, the description of a process is captured in a separate entity-type, Process; the location of a process can be determined from other relationships in which Process participates; if the system being described is operational, use the relationship-type Subroutine-Updates-Element instead.

  - **Frequency-of-Update** -- you can define this relationship attribute-type to describe the frequency with which the process is expected to update the element.

- **Record-Contains-Element** -- a predefined relationship-type; for an operational system that uses records in files or databases to support data, this relationship-type permits the user to identify the record(s) in which the element is located; the description of a record is captured in a separate entity-type, Record; the location of a record, in turn, can be captured in relationship-types such as File-Contains-Record and Table-Contains-Record.

- **Report-Contains-Element** -- you can define this relationship-type; for an operational system that uses reports to distribute information collected from files or databases, this relationship-type permits the user to identify the reports(s) in which the element is used; the description of a report is captured in a separate entity-type, Report; a report's location can be captured in relationship-types such as System-Generates-Report and User-Is-Responsible-For-Report.

- **Subroutine-Accesses-Element** -- you can define this relationship-type; for an operational system, this relationship-type permits the user to identify the subroutine(s) that access the data element without updating it; the description of a subroutine is captured in a separate entity-type, Subroutine; the location of a subroutine, in turn, can be captured in relationships such as Program-Contains-Subroutine; if the system being described is under development, you may want to use the relationship-type Process-Accesses-Element instead.

  - **Frequency-of-Access** -- you can define this relationship attribute-type to describe the frequency with which the subroutine accesses the element.

- **Subroutine-Updates-Element** -- you can define this relationship-type; for an operational system, this relationship-type permits the user to identify the subroutine(s) that update the data element; the description of a subroutine is captured in a separate entity-type, Subroutine; the location of a subroutine, in turn, can be captured in relationships such as Program-Contains-Subroutine; if the system is under development, you may wish to use the relationship-type Process-Updates-Element instead.

  - **Frequency-of-Update** -- you can define this relationship attribute-type to describe the frequency with which the subroutine accesses the element.

- Table-Contains-Element -- you can define this relationship-type; for an operational system that uses a relational DBMS to support data, this relationship-type permits the user to identify the database table(s) in which the element is located; the description of a relational table is captured in a separate entity-type, Table; the location of a table, in turn, can be captured with a relationship-type such as Database-Contains-Table.

Other metadata types useful for data element standardization are predefined in the Minimal and Basic Functional Schemas. Users may define any number of additional metadata types in the IRD schema.

Data element standardization is much easier to perform at the metadata level, with an IRD, rather than at the data level in a database. Since operational databases contain a minimum of self-descriptive information and contain a considerable amount of operational data, data element standardization in a database is awkward. The limited data dictionary facilities built into many DBMSs do not provide adequate support for thorough data element standardization.

Since the IRDS is designed to support metadata applications, such as data element standardization, metadata definition and analysis of this type can be performed more effectively with the support of an IRD.

An organization's IRD standards and conventions document should include a set of procedures for standardizing data elements. Such procedures will specify how IRD users should find and correct unintentional synonyms in data element names, and how to find and correct unintentional errors in data element definitions, descriptions, and assigned formats. The standards and conventions document should also provide users with procedures for using and maintaining data elements with the IRD.

4.7 Ensure IRD Security

An organization should establish conventions for assigning security permissions. Conventions on assigning security permissions should include a description of what types of personnel should have what types of permissions to read, add, modify, or delete different types of metadata in the various views and life cycle phases defined for one IRD.

Conventions defining data security measures should define the types of personnel who can modify the schemas, and designate the security methods used in the various IRDs owned by the organization.

In addition to this general set of security conventions, organizations should maintain a document listing the assignment of security permissions of different types to particular individuals. Since the IRDS Security module can support security permissions down to the entity-level, these permissions may have to be highly specific, according to the needs of the organization. Once these security permissions have been defined, they should be implemented in the IRD. Security use and permissions should be reviewed regularly for effectiveness.

5.0 Creating an IRD Schema

This chapter provides a general framework of concepts and procedures to be used in designing a schema to support an Information Resource Dictionary (IRD) application.

5.1 IRD Schema Concepts

A schema provides a means of representing information in a database or data dictionary. The construction of the schema determines the **types** of information that can be captured in the database or data dictionary.

Much as data types must be declared at the beginning of a program, **metadata types** must be defined before beginning to use a data dictionary. While the IRDS provides a number of predefined metadata types in its Minimal and Basic Functional Schemas, the IRDS user must consider if any additional metadata types are needed for a particular application.

As data types must be specified appropriately in each program, metadata types must be specified appropriately in each IRD schema. The structure of the IRD **schema** determines the types of information that the user can represent in the IRD. Because of this significance, the IRD user should be interested in the definition of the schema.

Unlike many data dictionary systems, the IRDS provides a fully **extensible schema**. This schema extensibility gives users nearly complete flexibility in representation. The benefit of this extensibility also imposes a burden, however, in that the user must be responsible for the structure of the schema.

A number of basic schema structures are provided in the Minimal Schema and the Basic Functional Schema, described generally in Chapter 3 of this paper and in greater detail in [GOLD88a]. These schema "starter sets" are intended to support the functionality of the IRDS and help organizations begin to use the IRDS.

While IRDS users will want to continue to use **aspects** of the Minimal and Basic Functional Schemas in many advanced IRD applications, the schemas provided with the IRDS are designed to satisfy only a **limited subset** of users' needs. IRDS users should plan to expand and redefine their IRD schemas as appropriate for their unique applications.

When an organization wants to use the IRDS to capture **life cycle** system development information, users should use **one IRD partitioned into phases.** The phase-related partitions of the IRD should be defined with **separate but related schemas.** These phase-related schemas can be defined at once or incrementally.

During the IRD schema definition process, users should consider the **traceability** they wish to capture from phase to phase. Users of the IRDS Life Cycle Phase module will want to be able to define **cross-referencing relationships** between entities of different phases. To establish relationships between phase-related entities, the user must define the appropriate meta-relationship structures in the IRD phase-related schemas. The procedures for defining cross-referencing relationships across phases will be discussed in detail in a subsequent publication.

A **subschema** is a subset of a schema that defines the structure of a database or data dictionary. Also known as an "external schema", a subschema is usually a **part of the schema** that supports a user group's **view**, or partial perspective of a database or data dictionary. Since an IRD can support a number of user groups that have different views of the dictionary, each IRD schema can have many subschemas.

While the IRDS standard does not now specify full subsetting capabilities to derive a subschema from a schema, the IRDS does provide users a capability to **subset a view.** Through the view facility, users can access any individual view and find the subschema, the portion of the schema, that supports it.

5.2 Top-Down Planning for IRD Use

The first procedure for any IRD application should be to define the **purpose** of your planned metadata collection. For instance, users should decide if the IRD will be used to support data element standardization for several systems, to support logical database design for one system, or to support all life cycle phases for one system.

The next step should be to define what **results** or **goals** must be accomplished to achieve this **purpose** for the IRD. For instance, what specific **reports, queries, information models, data flows,** or **documents** should result from the information you plan to capture in the IRD? For each of these planned results, what types of metadata must be represented?

If your purpose is to support **Strategic Systems Planning,** what results do you want to achieve with the IRD? Will IRD output lists be sufficient to represent the **Business Model** and **Global Data Model**? Or must specific IRD output be generated within a document, such as a Concept of Operations?

If specific IRD output must appear within a document, what particular types of metadata should be retrieved from the IRD for that document? Should IRD output be converted into **input for a document generator** that will insert metadata in appropriate portions of document text?

When a **top-down approach** is used by defining the project's **purpose** and **results** before beginning an IRD application, a user group can plan an IRD schema to be sufficient to support its purpose and goals. If you do not plan ahead for the purpose and goals of your IRD project, however, your organization may find it necessary to redesign the IRD schema several times throughout IRD development.

The IRDS fully supports schema modification. Once metadata has been defined and an IRD is in use, however, schema redesign can be time-consuming. Since schema modification to a functioning dictionary may result in metadata loss or reentry, top-down IRD project planning is recommended before beginning IRD schema definition.

5.3 Metadata Model Design

These preliminary steps should be performed to develop a metadata model before you begin to devise your IRD schema or application. The metadata application problem statements and structures that you develop in this procedure will determine how your IRD schema should be defined.

5.3.1 Isolate Metadata Subjects

The first step in developing an IRD is to isolate the one or more subjects of your immediate metadata collection. For instance, the **subject** of your data collection might be the development of a global data flow and global data model for Strategic Systems Planning.

As your organization learns how to use the IRDS, subject isolation may be performed initially by separate user groups. Once your organization has gained greater familiarity with the IRDS, different user groups that are working on the same system development project can be coordinated to combine their subject areas into one IRD, or a set of related IRDs. Experienced user groups should be able to combine a number of subjects effectively in one IRD, with each user group referring to one or more **views** within a **phase partition** of the dictionary.

5.3.2 Develop Problem Statements

The second step is to define a few representative analysis examples or **problem statements** for each subject. A problem statement may take the form of one or more sketches of high-level data flows or a generalized global data model. Define the problem statements in sufficient detail that they provide you with enough information to cover the analysis procedures of each subject.

To avoid modifications to the IRD schema as an IRD application progresses, users should develop a comprehensive problem statement for the current life cycle phase. A comprehensive problem statement, or set of problem statements, represents all the metadata types needed to support the information to be developed during a particular IRD life cycle phase application.

5.3.3 Classify Entities and Attributes

Working from your analysis sketches, translate your examples into categories of metadata that can be represented as **entities**, **relationships**, and **attributes**. Since it can be difficult to distinguish which objects in your examples should be entities, relationships, and attributes, some guidance on Entity-Relationship-Attribute modeling is provided in this section.

5.3.3.1 Entity-Type Classification

Select objects from each example that resemble **nouns**, and designate these as **entities**. If you plan to collect descriptive information (i.e., attributes) about a noun-like metadata object, that object is probably an entity. For example, some entities for an information system could be NETWORK-INTERFACE-DEVICE, FRONT-END-PROCESSOR, PAYROLL-SYSTEM, or WEEKLY-STATUS-REPORT. If you plan to use a metadata object to describe an entity, that object is an attribute.

Entity names can be the same names as you used in your example, as modified by your **naming conventions**. You should consider naming conventions early in your project, to standardize your name usage from the beginning [NEWT87]. Naming conventions are described briefly later in this chapter.

Entities should be examined to determine if they are all of one type or of different types. If you have difficulty determining their **entity-types**, consider if the set of entities can be divided into one or more general categories.

Establish a number of categories of entities, and decide how the entities can be assigned to these categories. Consider these categories as candidates for your entity-types. **Entity-type names** should be defined for each category, such as HARDWARE-ITEM, SOFTWARE-ITEM, SYSTEM, or REPORT.

Examples of entity-types are illustrated in Figure 6, in terms of the Entity-Relationship-Attribute model used with the IRDS. Entity-type names can be chosen from those defined in the Minimal and Basic Functional Schemas, or you **may define your own entity-types.** Each entity-type can refer to many entities, while each entity derives from one entity-type. Since these entity-types will be **the first major organizing factor of your IRD,** entity-type names should be used consistently with the same meanings. The **meaning or purpose** of each entity-type can be stored in the IRD schema as a value of the **meta-attribute-type "purpose"**, which has been predefined in the Minimal Schema.

5.3.3.2 Attribute-Type Classification for Entities

If you have information that you will want to use to describe these entities, then you want to define one or more **attributes.** A metadata object can be recognized as an attribute if it is descriptive, and if, at the metadata level, it has only an identifier and a metadata value. Attributes can be used to describe either entities or relationships. Attributes that are used like **adjectives** describe **entities**, and attributes that are used like **adverbs** describe **relationships.** Individual attribute values are supported by the definition of attribute-types. Each attribute-type can be used to represent many attributes (i.e., or many attribute values), while each attribute must have only one attribute-type.

An **attribute-type** must be defined before individual attribute values of this type can be assigned. To find appropriate attribute-types, locate the attribute values in your example that describe entities, divide these attribute values into categories, and devise names for these categories. These category names are the **attribute-types** that will be used to describe the entity-types in your application.

For an **entity-type** of **PROGRAM**, for example, a useful **attribute-type** might be **LANGUAGE**, for which the attribute value is the name of a programming language, such as **"PASCAL"**. Another useful attribute-type might be **LENGTH**, for which the **attribute** value would be the number of lines of code in a particular program, such as **"16"**. While the **attribute value** of "PASCAL" may not need any further explanation, the attribute value of "16" has little meaning without the **unit of measure**, which in this case is "lines of code." Capturing the unit of measure for attribute-types is discussed briefly in Chapter 6.

Example Entity-Relationship-Attribute Model for an Information Resource Dictionary

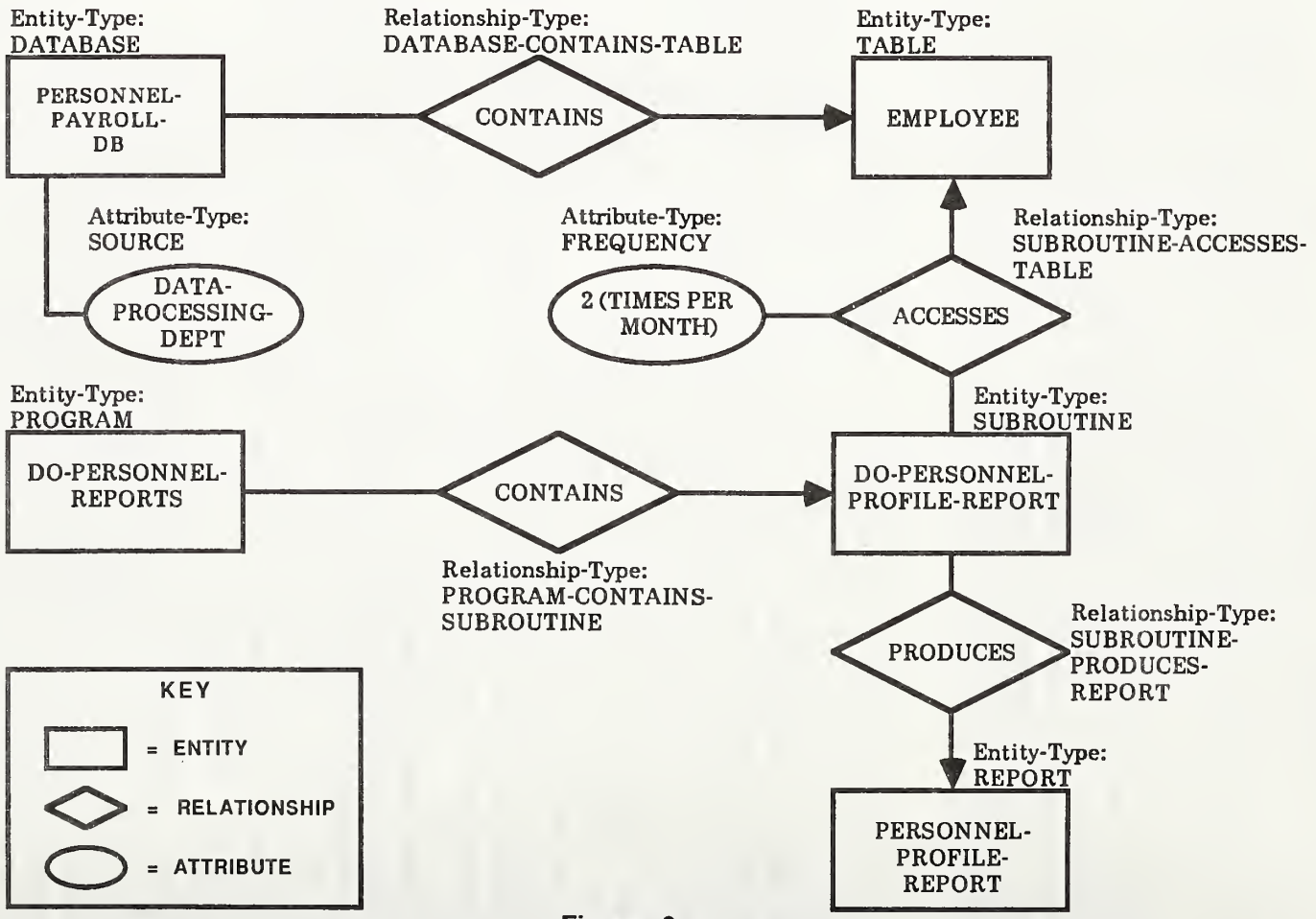


Figure 6

5.3.3.3 Associating Attribute-Types with Entity-Types

After the definition of attribute-types, the user should decide which attribute-types will be assigned to which entity-types. An attribute-type may be used to describe more than one entity-type. As entity-types and attribute-types are in the process of being developed and associated, different entity-types may share the same set of attribute-types. When an IRD schema is complete, however, each entity-type should have its own unique set of attribute-types.

When the attribute-types needed for entities are described, if a number of **different entity-types** share the **same set of attribute-types**, these entity-types should be **combined to form one, more general entity-type.** This elimination of redundancy assists in normalizing an IRD.

5.3.4 Classify Relationships and Attributes

Relationships between entities provide the second major organizing factor for your dictionary. You should consider how the entities in your problem statement interrelate, and define relationships for your problem statement at this point, if you have not done so previously.

5.3.4.1 Relationship-Type Classification

The user should define additional relationship-types to structure the relationships to be used in an IRD application. A **relationship-type** is based on the concept that the **central verb** of the **relationship-class-type** (e.g., CONTAINS) joins **two entity-types** (PROGRAM-CONTAINS-SUBROUTINE). Figure 6 illustrates other examples of relationship-types and shows one relationship with an attribute-type association.

Relationships drawn from your problem statement should be categorized to obtain their **relationship-class-types** (central verbs) and **relationship-types** (association of entity-types). A relationship-type is composed of:

Entity-type - Relationship-class-type - Entity-type

For instance, if your example problem is a **data flow diagram** in which a relationship shows data flowing from one entity to another, a **relationship-class-type** such as **FLOWS-TO** or **UPDATES** can be defined. If the two entity-types involved are **DATA-SET** and **DATA-STORE**, a **relationship-type** should be defined such as **DATA-SET-FLOWS-TO-DATA-STORE**, or **DATA-SET-UPDATES-DATA-STORE.**

5.3.4.2 Characteristics of IRD Relationship Structures

The IRDS recognizes only **binary** relationships, which associate two entities. The IRDS does not recognize ternary relationships, which associate three entities. If you wish to represent ternary relationships in your IRD application, they can be represented as entities.

At present, the relationships represented in an IRD are **unidirectional**, not reciprocal. For example, if you define relationships based on the relationship-type DATABASE-CONTAINS-TABLE, the IRDS does **not automatically define** or maintain a reverse relationship-type, TABLE-IS-CONTAINED-IN-DATABASE. While such a reciprocal relationship capability would be useful, it has not been specified as part of the current standard.

Although the IRDS will support the definition of the reverse relationship-type and relationships based on it, the IRDS does **not recognize** any association between the two relationship-types or relationships based on them. To avoid possible **data integrity errors** that can result when one relationship is modified but the reverse relationship is not, IRD users should define only unidirectional relationship-types and relationships.

In the event that reciprocal relationships are critical to the success of your IRD application, your organization's IRD Administrator should **devise procedures to protect the integrity** of the dictionary. In this case, the IRD Administrator should enforce the use of batch files for metadata creation and updates that **must** contain the appropriate reciprocal commands for the coordinated definition, modification, or deletion of such paired reciprocal relationships.

5.3.4.3 Attribute-Type Classification for Relationships

The IRDS supports the association of **attribute-types** with **relationship-types** in the IRD schema, in much the same way that attribute-types are associated with entity-types. This schema structure permits users to define attributes to describe relationships.

Find the attributes in your example that describe **relationships**, and classify these attributes into categories of **attribute-types**. For a **relationship-class-type** of **COMMUNICATES-WITH**, for example, that describes an association between two entities of entity-type **SITE**, a useful **attribute-type** might be **THROUGHPUT**. The **attribute value** for THROUGHPUT would be a number such as "80" with the unit of measure being the "number of megabytes per second" to be supported in communication between site A and site B. The representation an attribute-type's unit of measure is discussed briefly in Chapter 6.

Some relationship attribute-types may be critical to the success of your IRD application. For instance, in any Logical **Database Design** application, **connectivity** attribute-types should be used to describe all relationships. Connectivity describes the occurrence of entity instances for each relationship, such as one-to-one, one-to-many, or many-to-many entity instances in a relationship.

While schema structures to support connectivity are not predefined by the IRDS, connectivity can be represented in an IRD with the IRDS extensible schema. Users can define additional attribute-types to describe relationship connectivity as appropriate for Logical Database Design applications.

5.3.4.4 Associating Attribute-Types with Relationship-Types

After the definition of attribute-types to describe relationships, the user should decide which attribute-types will be assigned to which relationship-types. An attribute-type may be used to describe more than one relationship-type. Each relationship-type, however, should have its own **unique set** of attribute-types.

Once all the attribute-types needed for the relationships in your application are described, you may find that a number of **different relationship-types** share the **same set of attribute-types.** You should consider combining these relationships-types **to form one, more general relationship-type**, if this combination can be accomplished without loss of meaning. Such an elimination of redundancy assists in normalizing an IRD.

5.4 IRD Schema Description

After Entity-Relationship-Attribute modeling is completed for your problem statement, and you have determined which IRD to use, you are ready to define the **life cycle phase**, **views**, and **schema structures** appropriate to your IRD application. A new IRD can be created for your application schema, with your metadata added to a life cycle phase partition, accessed through a view. Or a suitable existing IRD can be used, with your metadata defined through a view within a life cycle phase partition. The predefined Uncontrolled phase can be used, or a new user-defined phase can be added as part of the schema.

To create your IRD, you must **either** use a **predefined** schema or **define your own schema** as necessary. If your IRDS includes the Basic Functional Schema, and if your application **does** not **exceed** the structure of the Minimal and Basic Functional schemas, you can begin adding your metadata.

To use an **extended schema** with additions to the predefined Minimal and Basic Functional schemas, follow the procedures described below to define a schema sufficient to represent the problem statement for your application. This section briefly illustrates the commands necessary to define IRD schema additions. Further information on the IRDS command language, including schema definition commands, is available in [GOLD88b].

The IRD schema must be defined in the appropriate schema description terms. If you will look back to Figure 5, you will see most of the appropriate terms listed in the top layer, the IRD Schema Description layer. The primary terms used in IRD schema description either begin with **"meta"** such as meta-entity, meta-relationship, and meta-attribute, or end in **"type"** such as entity-type, relationship-type, and attribute-type.

With the command language interface, each schema creation command line starts with "add meta," such as "add meta-entity." Using the **"add meta-entity"** command, represent the names for each category of entity, attribute, and relationship as **entity-type**, **attribute-type**, and **relationship-type**. With these commands, you define your IRD schema to support the metadata of your problem statement in terms of entities, attributes, and relationships raised to a higher, "meta" level.

5.4.1 Entity-Type Definition

Each entity that you add as metadata in your dictionary must be defined as having a particular entity-type. To define a new **entity-type** in the schema, the command format is:

Add meta-entity **entity-type-name** meta-entity-type = entity-type;

If you want to use **entity-types** such as PROGRAM, SUBROUTINE, TABLE, FIELD, and VARIABLE to capture information about a program that uses a relational DBMS to access tables and fields, you must define the commands:

Add meta-entity **SUBROUTINE** meta-entity-type = **entity-type**;
Add meta-entity **VARIABLE** meta-entity-type = **entity-type**;
Add meta-entity **DATABASE** meta-entity-type = **entity-type**;
Add meta-entity **TABLE** meta-entity-type = **entity-type**;
Add meta-entity **FIELD** meta-entity-type = **entity-type**;

The entity-type PROGRAM, which could have been defined with commands like those above, is predefined for you in the Basic Functional Schema, so you will not need to redefine it.

5.4.2 Relationship-Type Definition

After you have defined the entity-types you plan to use, as shown above, you can use the "add meta-entity" command again to define any **relationship-types** you may want to establish between any two entity-types. New relationship-types are defined with the command format:

```
Add meta-entity relationship-type meta-entity-type =
relationship-type;
```

To define the relationship-types of DATABASE-CONTAINS-TABLE, TABLE-CONTAINS-FIELD, SUBROUTINE-ACCESSES-TABLE, SUBROUTINE-ACCESSES-FIELD, and SUBROUTINE-RETURNS-VARIABLE, you must issue the commands:

```
Add meta-entity DATABASE-CONTAINS-TABLE meta-entity-type
= relationship-type;

Add meta-entity TABLE-CONTAINS-FIELD meta-entity-type
= relationship-type;

Add meta-entity SUBROUTINE-ACCESSES-FIELD meta-entity-type
= relationship-type;

Add meta-entity SUBROUTINE-RETURNS-VARIABLE meta-entity-type
= relationship-type;
```

5.4.3 Optional Relationship-Class-Type Definition

The optional relationship-class-type definition may be used to help you structure your schema. New relationship-class-types are defined according to the format:

```
Add meta-entity relationship-class-type-name
meta-entity-type = relationship-class-type;
```

For example, if you wanted to use the relationship-types of SUBROUTINE-ACCESSES-TABLE, or SUBROUTINE-ACCESSES-FIELD, you can define the relationship-class-type of **ACCESSES.** Similarly, if you want to use the relationship-type of SUBROUTINE-RETURNS-VARIABLE, you can define the relationship-class-type of **RETURNS.** While relationship-class-types may seem cumbersome, they are helpful in organizing an application.

To specify the optional **relationship-class-types**, use the command form:

Add meta-entity **relationship-class-type-name**
meta-entity-type = **relationship-class-type;**

The following examples show how to define a relationship-class-type in your IRD schema:

Add meta-entity **ACCESSES** meta-entity-type =
relationship-class-type;

Add meta-entity **RETURNS** meta-entity-type =
relationship-class-type;

The relationship-class-type CONTAINS is predefined for you in the Basic Functional Schema, from which you can later draw relationship-types, such as DATABASE-CONTAINS-TABLE and TABLE-CONTAINS-FIELD.

Following the specification of relationship-class-types and relationship-types, you should associate the two with the **"add meta-relationship"** command.

5.4.4 Relationship-Type Defined as a Relationship-Class-Type

If you have defined relationship-class-types, then you must associate the appropriate relationship-types with these new relationship-class-types. Such an association allows the IRDS to know that a relationship-class-type refers to one or more relationship-types. The command for this association follows the form:

Add meta-relationship **relationship-type** member-of
relationship-class-type;

The following examples show how relationship-type membership in relationship-type-classes can be defined in your IRD schema:

Add meta-relationship **TABLE-CONTAINS-FIELD**
member-of **CONTAINS;**

Add meta-relationship **DATABASE-CONTAINS-TABLE**
member-of **CONTAINS;**

```
Add meta-relationship SUBROUTINE-RETURNS-VARIABLE
member-of RETURNS;

Add meta-relationship SUBROUTINE-ACCESSES-FIELD
member-of ACCESSES;
```

5.4.5 Relationship-Type Positional Definition

For each relationship-type that you have added to your schema, as illustrated above, you must establish the **position** of each **entity-type** in that **relationship-type**. Similarly, if you have defined new attribute-group-types, you must establish the position of each component attribute-type in the group.

Use the **"add meta-relationship"** command to define the positional placement for each entity or attribute-type. This command is necessary for the IRD to determine whether the entity-type is in **position = 1** at the beginning of the relationship, or whether the entity-type is in **position = 2** at the end of the relationship. The format of the command to establish entity-type placement in relationship-type statements is:

```
Add meta-relationship relationship-type
connects first-entity-in-relation position = 1;

Add meta-relationship relationship-type
connects second-entity-in-relation position = 2;
```

For the relationship-types **TABLE-CONTAINS-FIELD**, **SUBROUTINE-ACCESSES-TABLE**, and **SUBROUTINE-RETURNS-VARIABLE**, the following meta-relationship commands are needed to establish entity placement:

```
Add meta-relationship TABLE-CONTAINS-FIELD
connects TABLE position = 1;

Add meta-relationship TABLE-CONTAINS-FIELD
connects FIELD  position = 2;


Add meta-relationship SUBROUTINE-ACCESSES-TABLE
connects SUBROUTINE position = 1;

Add meta-relationship SUBROUTINE-ACCESSES-TABLE
connects TABLE  position = 2;
```

```
Add meta-relationship SUBROUTINE-RETURNS-VARIABLE
connects SUBROUTINE  position = 1;

Add meta-relationship SUBROUTINE-RETURNS-VARIABLE
connects VARIABLE  position = 2;
```

5.4.6 Attribute-Type Definition

Categories of additional **attributes**, beyond those defined in the Minimal and Basic Functional Schemas, must be defined as **attribute-types** with the "add meta-entity" command.

The **Minimal Schema** provides a number of useful attribute-types and attribute-group-types that will probably be used in most IRD applications. For example, the Minimal Schema provides users with the attribute-types **NUMBER-OF-TIMES-MODIFIED, LAST-MODIFIED-BY**, as well as the attribute-group-type **DATE-TIME-LAST-MODIFIED**, which includes the attribute-types **SYSTEM-DATE** and **SYSTEM-TIME**. All of the attribute-types of the Minimal Schema, such as these, are automatically system maintained.

These attribute-types and attribute-group-types in the IRD schema permit the IRD to record, for example, the number of times an entity has been modified, the person who made the last modification, and the time/date of that modification.

The **Basic Functional Schema** provides a number of attribute-types, such as **FREQUENCY**, that will prove useful in your IRD applications. The Basic Functional Schema also predefines for you an attribute-group-type **IDENTIFICATION-NAMES**, which includes the attribute-types **ALTERNATE-NAME** and **ALTERNATE-NAME-CONTEXT**. With this schema structure, you will be able to define attributes for ALTERNATE-NAME, or the alias names of an entity, and attributes for ALTERNATE-NAME-CONTEXT, such as the department or system where that alias is used.

You may define attribute-types beyond those provided by the Minimal and Basic Functional schemas. The format for the definition of a new attribute-type is:

```
Add meta-entity attribute-type-name meta-entity-type
= attribute-type;
```

Many of the attribute-types users will want have predefined in the Minimal and Basic Functional schemas, but you may find that you want to add additional ones. For example, you may find the attribute-type **ADDED-BY**, provided by both the Minimal and Basic Functional schemas, useful in designating the person who added a particular schema or metadata item to the dictionary.

If you wanted to use an **additional** attribute-type to describe the **department** that added an entity, however, you will need to define a new attribute-type. You could define the attribute **SOURCE**, for instance, to designate the department or other **organizational unit** responsible for defining metadata items. To define SOURCE as a valid attribute-type that you will be able to use later with entities and relationships, use the command:

Add meta-entity **SOURCE** meta-entity-type = **attribute-type**
with purpose = "Department that defined this metadata";

If you want to define additional **attribute-group-types**, you will use a similar command format:

Add meta-entity **attribute-group-type-name** meta-entity-type = **attribute-group-type;**

For instance, you might want to be able to describe metadata of the relationship-type **SUBROUTINE-ACCESSES-TABLE** with a **permitted access range** that will define the limits of how often a table should be accessed. If you wanted to be sure that a table was accessed for updates at a certain minimum interval and that a table used by many departments was not accessed for update too often for the current physical database design, you might want to define these access limits for specific tables. This range could be used to trigger a warning message when the limits are passed.

To support metadata describing the permitted limits for a subroutine to access a table, you could define an **attribute-group-type** with the name of **PERMITTED-ACCESS-RANGE**, to include the attribute-types **MINIMUM-ACCESS-VALUE** and **MAXIMUM-ACCESS-VALUE.** Since you will probably want the metadata values to be numeric, your standards and conventions document should include information about the method of measurement, such as "per day," or "per hour." To define such an attribute-group-type and its component attribute-types, use the following commands:

Add meta-entity **PERMITTED-ACCESS-RANGE** meta-entity-type = attribute-group-type;

Add meta-entity **MINIMUM-ACCESS-VALUE** meta-entity-type = attribute-type;

Add meta-entity **MAXIMUM-ACCESS-VALUE** meta-entity-type = attribute-type;

5.4.7 Attribute-Type Definition of Attribute Value Format

Depending on the schema definition for each **attribute-type**, attribute values can be added to an IRD in a number of different **formats.** The format for **attribute values** of any attribute-type are specified in the schema with the **FORMAT meta-attribute** of the attribute-type meta-entity. The permitted values for the FORMAT of an attribute-type are STRING, TEXT, INTEGER, REAL, DATE, and TIME. The **default value** for FORMAT is **STRING**, which permits the options of: (1) a single term with no embedded blanks; (2) a string with no embedded blanks, in which the terms of the string are separated by hyphens; or (3) a quoted string in which embedded blanks are permitted.

To define the appropriate FORMAT for a particular attribute-type, the user can simply include the FORMAT meta-attribute into the "add meta-entity" command shown above:

Add meta-entity **attribute-type-name** meta-entity-type = **attribute-type** with **FORMAT = integer;**

The same form would apply for attribute-types to be associated with an attribute-group-type. Since attribute values will **not** be directly assigned to attribute-group-types, an attribute-group-type **does not** directly receive the FORMAT meta-attribute-type. The command would appear as:

Add meta-entity **PERMITTED-ACCESS-RANGE** meta-entity-type = **attribute-group-type;**

Add meta-entity **MINIMUM-ACCESS-VALUE** meta-entity-type = **attribute-type** with **FORMAT = integer;**

Add meta-entity **MAXIMUM-ACCESS-VALUE** meta-entity-type = **attribute-type** with **FORMAT = integer;**

If the attribute-types **have already been defined,** then the user should **modify** them, not define them again. To add the **FORMAT** meta-attribute for attribute values to attribute-types that have previously been defined, use the **"modify meta-entity"** command form:

Modify meta-entity **MINIMUM-ACCESS-VALUE** with **FORMAT = integer;**

Modify meta-entity **MAXIMUM-ACCESS-VALUE** with **FORMAT = integer;**

5.4.8 Attribute-Group-Type Positional Description

If you have defined attribute-group-types, you must also define the **positions** of the **attribute-types** that you want to include in those groups. Attribute-types are associated with attribute-group-types with an "add meta-relationship" command. The format of the command to **associate** an **attribute-group-type** with its **component attribute-types** is:

Add meta-relationship **attribute-group-type-name** contains **first-attribute-type** position = 1;

Add meta-relationship **attribute-group-type-name** contains **second-attribute-type** position = 2;

To associate the attribute-group-type PERMITTED-ACCESS-RANGE with its component attribute-types, use the following commands:

Add meta-relationship **PERMITTED-ACCESS-RANGE** contains **MINIMUM-ACCESS-VALUE** position = 1;

Add meta-relationship **PERMITTED-ACCESS-RANGE** contains **MAXIMUM-ACCESS-VALUE** position = 2;

5.4.9 Attribute-Type Association with Entity-Type

When you want to use particular attribute-types with particular entity-types, you must define this association in the IRD schema. The Minimal and Basic Functional schemas provide this **association** of **some** existing attribute-types with **some** existing entity-types. See [GOLD88a] for definition of which existing attribute-types have predefined associations with which existing entity-types.

When you **add entity-types**, you **must associate** each new entity-type with any attribute-types you want to use to describe it. Since the attribute-types provided in the Minimal Schema are automatically maintained, you do not need to be concerned with those. However, if you want to use **predefined attribute-types from the Basic Functional Schema** to describe **additional entity-types** that you have defined, you **must associate** those attribute-types with your new entity-types. The format for the command that establishes this association is:

Add meta-relationship **entity-type-name** contains **attribute-type-name**;

If you want to be able to define metadata showing what departmental sources are responsible for which particular tables, fields, databases, programs, and subroutines, you should define the following commands:

Add meta-relationship **TABLE** contains **SOURCE**;
Add meta-relationship **FIELD** contains **SOURCE**;
Add meta-relationship **DATABASE** contains **SOURCE**;
Add meta-relationship **PROGRAM** contains **SOURCE**;
Add meta-relationship **SUBROUTINE** contains **SOURCE**;

Just as you must associate attribute-types with entity-types, so must you associate attribute-group-types with the appropriate entity-types. The command that establishes the association of attribute-group-types with entity-types is:

Add meta-relationship **entity-type-name** contains
attribute-group-type-name;

If you want to be able to describe tables and databases with metadata for the predefined attribute-group-type IDENTIFICATION-NAMES, which includes the attribute-types ALTERNATE-NAME and ALTERNATE-NAME-CONTEXT, define the following commands:

Add meta-relationship **TABLE** contains **IDENTIFICATION-NAMES**;
Add meta-relationship **FIELD** contains **IDENTIFICATION-NAMES**;

5.4.10 Attribute-Type Association with Relationship-Type

To use attribute-types with a particular relationship-type, you must define this association in the IRD schema. The Minimal and Basic Functional schemas associate a few predefined attribute-types with a few predefined relationship-types. See [GOLD88a] for definition of which existing attribute-types have predefined associations with which existing relationship-types.

When you **add new relationship-types**, you **must associate** each new relationship-type with any attribute-types to be used with it. You do not need to be concerned with the attribute-types predefined in the Minimal Schema, since those are maintained automatically. If you use **attribute-types predefined in the Basic Functional Schema** with **new relationship-types** that you have added, you must **associate** those attribute-types with your new relationship-types.

The command to associate an attribute-type with a relationship-type has the following format:

Add meta-relationship **relationship-type-name** contains **attribute-type-name;**

If you want to be able to define metadata showing how frequently particular tables and databases are accessed, using the attribute-type FREQUENCY, define the following commands:

Add meta-relationship **SUBROUTINE-ACCESSES-TABLE** contains **FREQUENCY;**

Add meta-relationship **PROGRAM-ACCESSES-DATABASE** contains **FREQUENCY;**

The association of attribute-group-types with relationship-types follows a similar pattern.

Add meta-relationship **relationship-type-name** contains **attribute-group-type-name;**

5.5 Life Cycle Phase Partitioning

Life cycle phase **partitions** can be used to organize the IRD schema and metadata into different **phases** within an IRD. The schema and metadata of an IRD **must** be defined in terms of a life cycle phase partition.

5.5.1 IRDS Life Cycle Phase Concepts

The predefined IRDS **life cycle phases** are UNCONTROLLED-LIFE-CYCLE-PHASE, CONTROLLED-LIFE-CYCLE-PHASE, and ARCHIVED-LIFE-CYCLE-PHASE. The UNCONTROLLED-LIFE-CYCLE-PHASE corresponds to **systems development** metadata, the CONTROLLED-LIFE-CYCLE-PHASE corresponds to **system operations** metadata, and the ARCHIVED-LIFE-CYCLE-PHASE corresponds to **former system operations** metadata.

In addition to the **UNCONTROLLED-LIFE-CYCLE-PHASE**, any number of additional Uncontrolled phases can be user-defined for system development support, such as the STRATEGIC-SYSTEMS-PLANNING and REQUIREMENTS-DEFINITION phases. IRD metadata can **only be** defined in **user-defined Uncontrolled phases** such as these, or in the **UNCONTROLLED-LIFE-CYCLE-PHASE.**

Metadata can only be placed into the **CONTROLLED-LIFE-CYCLE-PHASE** through the **transfer** of metadata from the UNCONTROLLED-LIFE-CYCLE-PHASE or from one of the user-defined Uncontrolled phases. In turn, CONTROLLED-LIFE-CYCLE-PHASE metadata concerning current system operations can be **transferred** to the **ARCHIVED-LIFE-CYCLE-PHASE**, which is designed to record systems operations metadata that is out-of-date. Metadata can **only** be placed into the ARCHIVED-LIFE-CYCLE-PHASE is through this **transfer** of metadata from the CONTROLLED-LIFE-CYCLE-PHASE.

5.5.2 Life Cycle Phase Definition

Additional life cycle phase partitions can be defined within an IRD to represent the phases of the system development life cycle. **Only** additional **Uncontrolled** life cycle phases can be defined; no other Controlled or Archived phases can be defined. These additional Uncontrolled phases are defined in the schema as meta-entities of the type **IRD-Partition.**

Additional Uncontrolled life cycle phase partitions can be defined with the following command format:

Add meta-entity **PHASE-NAME** meta-entity-type = **IRD-Partition;**

The following are examples of the command to define new Uncontrolled life cycle phases:

Add meta-entity **STRATEGIC-SYSTEMS-PLANNING**
meta-entity-type = **IRD-Partition;**

Add meta-entity **REQUIREMENTS-DEFINITION**
meta-entity-type = **IRD-Partition;**

While there are additional IRD schema definition commands, such as those for schema modification and deletion, the major terms of your initial IRD schema have been described in this chapter. Once an appropriate schema has been defined, users can go on to add metadata to the IRD.

6.0 Creating an IRD Application

When the schema is specified for your IRD, you can begin to add metadata to the dictionary in terms of the entity-types, attribute-types, and relationship-types that have been defined. The schema of your IRD can combine the predefined IRDS Minimal and Basic Functional Schemas with your user-defined schema definition commands.

6.1 IRD View Definition and Access

A **view**, the user's doorway into a dictionary or a phase, permits the user to access either a limited portion of an IRD or an entire IRD, depending on how the view is defined.

Each view must be defined in terms of the life cycle phase **partition** in which it is used. View and life cycle phase facilities are supported by the Minimal Schema, so any conforming IRDS can support the definition of views and phases.

Every IRD user must be assigned at least one **permitted view**, usually by the IRD Administrator. The user or the IRD Administrator must also assign the user at least one effective-view or default view, that corresponds to one of the user's permitted views. The **effective-view** is the **user's current operating view** of an IRD, which may be changed according to the user's wish and view permissions.

The user's view permissions and effective-view may be relatively transparent to the user, or the user's access to information in the dictionary can be restricted. Restrictive view permissions can be defined with the **Security Facilities** optional module to provide greater protection for an IRD's metadata and schema.

6.1.1 View Definition within a Life Cycle Phase

The IRD Administrator, or another user who has been granted administrator privileges, can define views within life cycle phases. A view provides a doorway into an IRD life cycle phase through which the user can: (1) access the life cycle phase, (2) store metadata in the life cycle phase, and (3) retrieve metadata from the life cycle phase. An IRD life cycle phase **must always be accessed through a view.** Users should consider the life cycle phase and view in which metadata should be placed **before** defining the view and its associated metadata, since it can be time-consuming to transfer metadata from one phase to another.

Before defining a **view** within a particular **user-defined life cycle phase**, the IRD administrator must have previously defined that life cycle phase name. A view is associated with a phase by means of the **IRD-Partition-Name attribute** of the **IRD-View entity**. The metadata command format to define a **view** within a phase is:

```
Add entity VIEW-NAME entity-type = IRD-View
with IRD-Partition-Name = PHASE-NAME;
```

Examples of the command to define views within the user-defined REQUIREMENTS-DEFINITION phase are:

```
Add entity DATA-MANAGEMENT-VIEW entity-type = IRD-View
with IRD-Partition-Name = STRATEGIC-SYSTEMS-PLANNING;

Add entity DATA-PROCESSING-VIEW entity-type = IRD-View
with IRD-Partition-Name = STRATEGIC-SYSTEMS-PLANNING;

Add entity DATA-COMMUNICATIONS-VIEW entity-type = IRD-View
with IRD-Partition-Name = STRATEGIC-SYSTEMS-PLANNING;

Add entity DATA-CONVERSION-VIEW entity-type = IRD-View
with IRD-Partition-Name = STRATEGIC-SYSTEMS-PLANNING;
```

6.1.2 View Access Permissions

In order for a user to access a view, the user must be **associated** with that view. A user can be associated with any number of views. The IRD Administrator is the appropriate person to define views and grant users permission to access particular views. The IRD Administrator must have previously defined the appropriate **user names** and **view names** before issuing the command to associate a user with a view. The metadata command to **associate the user with a view** has the following format:

```
Add relationship USER-NAME user-has-IRD-view VIEW-NAME;
```

Examples of this command to associate a user with a view are:

```
Add relationship LAW user-has-IRD-view
DATA-MANAGEMENT-VIEW;
```

Add relationship **LAW** user-has-IRD-view
DATA-PROCESSING-VIEW;

Add relationship **QUINN** user-has-IRD-view
DATA-PROCESSING-VIEW;

6.1.3 The User's Effective-View

While a user may have access to many views, at any given time a user has one and only one **effective-view.** Either the IRD Administrator or the user can define the user's effective-view. Since a user may wish to change his or her effective-view relatively often, we will consider the effective-view command from the user's perspective.

The user does not need to identify him or herself, because the **user's login** to the IRD defines the user's name. The command the user issues to define his or her effective-view has the following format:

Set IRD view = **VIEW-NAME;**

An example of this command is:

Set IRD view = **DATA-COMMUNICATIONS-VIEW;**

The user cannot define a view as his or her effective-view unless the user has previously been **granted access permission** to that view. If the user wishes to change his or her effective view, the user just re-issues this command indicating another effective VIEW-NAME.

6.2 Metadata Definition Via Phases and Views

When defining metadata in the various life cycle phase partitions of an IRD, the user will only be able to **define** a **metadata item once.** Users cannot define the same metadata item, such as an entity, in more than one phase. If you wish to define an entity in more than one phase, you will have to **vary** the entity name slightly for each phase. Creating unique names for entities used in multiple phases can be supported through the use of the **variation name** facility.

For instance, the **variation name facility** can be used to vary the entity **EMP-NO**, first defined in the UNCONTROLLED-LIFE-CYCLE-PHASE, to **EMP-NO(R)** for the REQUIREMENTS-DEFINITION-PHASE, to **EMP-NO(F)** for the FUNCTIONAL-SPECIFICATION-PHASE, and **EMP-NO(L)** for the LOGICAL-DATABASE-DESIGN-PHASE, etc. Since each of these variation names is stored as a separate entity, you can define relationships between these different entities.

Cross-referencing relationships can also be established across phases. For example, if you wanted to show that the REQUIREMENTS-DEFINITION phase EMP-NO referred to the FUNCTIONAL-SPECIFICATION phase EMP-NO, you could define the relationship **EMP-NO(R) EMP-NO-REFERS-TO-EMP-NO EMP-NO(F)**, if your IRD schema is constructed to support such a relationship-type.

If **views have been defined** within the predefined UNCONTROLLED-LIFE-CYCLE-PHASE (i.e., instead of within a user-defined phase) the IRDS will read every metadata definition and place the new metadata into the **UNCONTROLLED-LIFE-CYCLE-PHASE** via the **user's effective-view.**

If **views have been defined and associated with user-defined life cycle phases**, the IRDS will read every metadata definition and place the new metadata into the **user-defined life cycle phase** with which the **user's effective-view** is associated.

3.5.4 View and Phase Status

If you are using the Panel Interface facility of the IRDS, your current **view and phase status** should be displayed in the **STATE** panel area. If you do not have the Panel Interface available to you, you can use the Session Status command to find your effective-view.

To list all the IRD views with which the user is associated, and to identify the user's **effective-view**, the user issues a **Session Status** command with the following format:

```
Status IRD-VIEW;
```

To find the **life cycle phase** with which your **effective-view** or another view is associated, use the command format:

```
Output IRD-NAME
select entities with access-name = VIEW-NAME
show attribute IRD-PARTITION-NAME;
```

6.2.2 Metadata Retrieval from Views and Phases

A set of output commands with selection criteria have been specified for the IRDS which allow the user to structure queries for output. While the subject of IRD output is too extensive to be covered adequately for the purposes of this guide, this section briefly discusses a few global output commands.

The global command to generate IRD output refers to the user's current **effective-view**, which is the user's default view. Depending on how the user's effective-view is defined by the IRD administrator, that view may permit access to the entire IRD or may restrict access to only part of it. Since the user's effective-view must be associated with a life cycle phase, the user's **retrieval of life cycle phase metadata** is determined by the **retrieval of view metadata.** In order to output all metadata within a **phase**, the user should retrieve all metadata from all the **views** defined for that phase.

A global command to generate IRD output by selecting all entities in the user's current **effective-view**, has the format:

```
Output IRD-NAME select all show all;
```

To issue a global command to output the contents of another **view** to which the user has access, the command format is:

```
Output IRD-NAME using-IRD-view = VIEW-NAME
select all show all;
```

A global command to output the combined contents of **several views** to which the user has access in one IRD has the format:

```
Output IRD-NAME using-IRD-view =   VIEW-NAME
                                   VIEW-NAME
                                   VIEW-NAME
                                   . . .
select all show all;
```

If the user wishes to issue a global command to output the combined contents of **all views** to which the user has been granted access permission within one IRD, the command format is:

```
Output IRD-NAME using-IRD-view = all
select all show all;
```

6.3 Entity Definition with Attributes

When the IRD schema has been defined, and user views have been defined and associated with phases, metadata can be added to populate your dictionary.

Metadata can be defined with the "add entity" and "add relationship" commands. Begin your metadata definition at the **entity level** with the "add entity" command, according to the following format:

```
Add entity entity-name entity-type = entity-type-name
      entity descriptive-name = full-name-that-you-provide
      with attribute-name = attribute-value,
      attribute-name = attribute-value,
      . . .
      attribute-group-name =
            (attribute-name = attribute-value,
             attribute-name = attribute-value),
             . . . ;
```

Any number of attributes or attribute-groups can be included in your entity definition. The use of the **"add entity"** command is illustrated in the examples that follow.

One potential application of an IRD is to document the **structure of computer programs** in information systems. For example, to document a system with many complex programs and subroutines, programmers can be required to define in an IRD the **purpose, primary actions, interactions, inputs,** and **outputs** for each **program**, **subroutine**, and **function**. Such an IRD would provide the **means to analyze the structure of a system** without having to read documentation embedded in the code.

If, for instance, you have one **program** that **converts data** from a data file into a format suitable to **update a relational database** running on a DBMS, you could use an IRD to document the structure of that program's **subroutines.** The entity-type **PROGRAM** is predefined for you in the Basic Functional Schema, with a set of attribute-types, so PROGRAM can be used as the basis of additional entities.

If this data conversion and database update system are used by a number of different departments within the organization, which have each provided subroutines to the program, system documentation should track the departmental **SOURCE** of each subroutine. Since different departments may call one subroutine by different names, system documentation should also track any **ALTERNATE-NAME** by which these subroutines are referred.

The commands to add metadata to an IRD for such an application could be:

```
Add entity UPDATE-PERSONNEL-PAYROLL-DB entity-type = PROGRAM
     entity descriptive-name =
     PREPARE-PERSONNEL-PAYROLL-DATABASE-UPDATES
     with SOURCE =  "PERSONNEL DEPT",
     IDENTIFICATION NAMES =
          (ALTERNATE-NAME= "PERSONNEL DB UPDATE PROGRAM",
           ALTERNATE-NAME-CONTEXT = "PERSONNEL DEPT");


Add entity DO-TABLES entity-type = SUBROUTINE
     entity descriptive-name =
     PREPARE-INPUT-FOR-TABLES
     with SOURCE = "DATA PROCESSING DEPT";


Add entity DO-FIELDS entity-type = SUBROUTINE
     entity descriptive-name =
     PREPARE-INPUT-FOR-FIELDS
     with SOURCE = "DATA PROCESSING DEPT";


Add entity PERSONNEL-PAYROLL-DB entity-type = DATABASE
     entity descriptive-name =
     PERSONNEL-AND-PAYROLL-DATABASE
     with SOURCE = "DATA PROCESSING DEPT",
     IDENTIFICATION-NAMES =
          (ALTERNATE-NAME = "PAYROLL DB",
           ALTERNATE-NAME-CONTEXT = "PAYROLL DEPT"),
          (ALTERNATE-NAME = "PERSONNEL DB",
           ALTERNATE-NAME-CONTEXT = "PERSONNEL DEPT");


Add entity EMPLOYEE entity-type = TABLE
     entity descriptive-name =
     EMPLOYEE-INFORMATION
     with SOURCE = "PERSONNEL DEPT",
     IDENTIFICATION-NAMES =
          (ALTERNATE-NAME = "EMP",
           ALTERNATE-NAME-CONTEXT = "PAYROLL DEPT"),
          (ALTERNATE-NAME = "EMPLOYE",
           ALTERNATE-NAME-CONTEXT = "PERSONNEL DEPT");


Add entity EMP-NO entity-type = FIELD
     entity descriptive-name =
     EMPLOYEE-IDENTIFICATION-NUMBER
     with SOURCE = "PERSONNEL DEPT";
```

The attribute-type SOURCE must be user-defined in the schema before metadata, such as that above, can be added. In addition to the few attribute-types used above, many other attribute-types are available through the Minimal and Basic Functional schemas. Descriptive-name is an entity name, not an attribute-type, so its value is NOT followed by a comma.

6.4 Attribute Value Formats

Depending on the schema definition for each **attribute-type**, attribute values can be added to an IRD in a number of different formats. The format for **attribute values** of any attribute-type are specified in the **schema** with the definition of the **FORMAT meta-attribute**. The permitted values for FORMAT are STRING, TEXT, INTEGER, REAL, DATE, and TIME.

The **default value** for FORMAT is **STRING**, which permits the options of: (1) a single term with no embedded blanks; (2) a string with no embedded blanks, in which the terms of the string are separated by hyphens; or (3) a quoted string in which embedded blanks are permitted.

Attribute value formats are discussed in the following section, and in Chapter 5 in terms of the IRD schema.

6.5 Relationship Definition with Attributes

After the appropriate **relationship-types** have been defined in your schema, and after the appropriate **entities** have been defined as metadata, **relationships** can be defined as metadata in your application. Relationships provide the basis for cross-referencing your metadata. Since a relationship is used to show the association between two entities of particular entity-types, the names of those **entities** and their **entity-types** are included in the relationship definition.

The relationship-type structure consists of the first **entity-type** to be associated, the **relationship-class-type**, and the second **entity-type** to be associated:

entity-type relationship-class-type entity-type

The relationship name structure consists of the first **entity name** in the relation, the full **relationship-type**, and the second **entity name** in the relation:

entity name relationship-type entity name

6.5.1 Defining Relationships

The **"add relationship"** command is used to enter relationship metadata into your IRD, as illustrated in the following examples. The following is the format of the command to add a relationship to associate two entities:

Add relationship entity-name relationship-type entity-name;

A relationship could be defined to show that a particular database, the PERSONNEL-PAYROLL-DB, **contains** a particular table, such as the EMPLOYEE table. Similarly, a relationship could be defined to show that table EMPLOYEE **contains** the element EMP-NO. Another relationship could be defined to show that the program UPDATE-PERSONNEL-PAYROLL-DB **contains** the subroutine DO-TABLES. The commands to add these relationships would be:

Add relationship PERSONNEL-PAYROLL-DB **DATABASE-CONTAINS-TABLE** EMPLOYEE;

Add relationship EMPLOYEE **TABLE-CONTAINS-ELEMENT** EMP-NO;

Add relationship UPDATE-PERSONNEL-PAYROLL-DB **PROGRAM-CONTAINS-SUBROUTINE** DO-TABLES;

6.5.2 Defining Attributes for Relationships

The **"add relationship"** command can also be used to enter add attribute metadata to relationships, as you define relationships for your IRD. The following is the format of the command to add a **relationship with an attribute:**

Add relationship entity-name relationship-type entity-name with **attribute-type-name = attribute-value;**

For example, a relationship could be defined to show that the subroutine DO-TABLES **accesses** the table EMPLOYEE, with an **attribute** defined to show that the relationship occurs with a certain FREQUENCY. The value of the attribute-type FREQUENCY can be defined as part of the "add relationship" command:

Add relationship DO-TABLES **SUBROUTINE-ACCESSES-TABLE** EMPLOYEE with **FREQUENCY** = 2;

The attribute-type FREQUENCY was not used to describe the relationship EMPLOYEE TABLE-CONTAINS-FIELD EMP-NO, since "frequency" does not apply to the relationship-class-type "contains."

6.5.3 Defining Attribute Units of Measure

The metadata value for the attribute-type FREQUENCY can be represented in the **FORMAT** of INTEGER, REAL, or STRING. If **STRING** is used as the FORMAT of attribute-type FREQUENCY, an attribute value of "2 TIMES PER WEEK" could be assigned to capture the unit of measure. While STRING permits the representation of the unit of measure, it also makes the numeric value more difficult to manipulate with an analysis program.

If **INTEGER** is used as the FORMAT of the attribute-type FREQUENCY, then the integer value of "2" is easier to work with, but the unit of measure may not be captured. Information about the unit of measure, such as "TIMES PER WEEK", could be maintained in the users' standards and conventions document. To capture **unit of measure** information, another alternative is to redefine FREQUENCY as an attribute-group-type associated with additional attribute-types such as FREQUENCY-VALUE and FREQUENCY-UNIT-OF-MEASURE. Other options for representing the unit of measure for an attribute-type will be covered in a subsequent publication.

7.0 Life Cycle Approach to IRD Applications

The IRDS can support metadata for all life cycle phases. As described in Chapter 3, the Core IRDS provides a basic life cycle phase capability that permits the user to define any number of system development life cycle phases. The IRDS also provides an optional module, the Extensible Life Cycle Phase Facility, that provides additional life cycle protection capabilities.

Life cycle phase partitions have been defined for the IRDS to support and organize the representation of the varied activities of information system development, operations, and maintenance. Without phase partitions, no one IRD would be sufficient to represent the spectrum of life cycle activities.

While it may not be practical to represent an entire system life cycle in a single IRD, the IRDS gives the user the flexibility to decide what portions of the life cycle should be represented together in one IRD at any particular time. Due to the capability of the IRDS to support relationships across phases, the IRDS permits the user to cross-reference metadata across phases. By cross-referencing entities across phases, the user can selectively trace the development of a system from concept to requirements, from requirements to specification, from specification to logical database design, etc.

Because of the intensive use of an IRD during system development, and the need for good IRD response time, the user may want to maintain only two life cycle phases within the same IRD at any one time. By working with any two phase partitions at a given time, the user can establish relationships across phases to establish traceability in development, and to verify that a requirement results in a function or data specification, and that a specification is reflected in the system design.

Function and data integration is at the heart of the system life cycle, as illustrated in Figure 7. Although it is not considered a life cycle phase, this type of information integration within and across phases is integral to the success of any systems development effort. The IRDS supports function and data integration, both within a phase and across phases.

7.1 Life Cycle Phase Applications

The IRDS supports function and data integration through the use of the Entity-Relationship-Attribute model. Since all functions and all data can be represented in an IRD life cycle phase as **entities** of different types, they can be interrelated through the use of **relationships**. Particular cross-referencing

relationship-types can be devised to support information integration. While entities must reside within a particular IRD life cycle phase, the IRDS permits relationships to span life cycle phases.

An IRD can support the following relationship categories for information integration within the same phase:

- Data entities can be associated with function entities, to show which function requires or uses what data.
- Data entities of one entity-type can be associated with data entities of another entity-type, to show hierarchical progression and integration in levels of detail for data description (i.e., "data leveling").
- Function entities of one entity-type can be associated with function entities of another entity-type, to show hierarchical progression and integration in levels of detail in functional description (i.e., "function leveling").
- Data entities can be associated with data entities of the same entity-type, to indicate a self-referencing data activity or recursion.
- Function entities can be associated with function entities of the same entity-type, to indicate a self-referencing function activity or recursion.

An IRD can support the following relationship categories for information integration across different phases:

- Data entities in one phase can be associated with function entities in another phase (or vice versa), to show which function requires or uses what data.
- Data entities in one phase of one entity-type can be associated with data entities in another phase of another entity-type, to show "data leveling" integration across phases.
- Function entities in one phase of one entity-type can be associated with function entities in another phase of another entity-type, to show "function leveling" integration across phases.
- Data entities in one phase can be associated with data entities in another phase of the same entity-type, to trace a self-referencing data activity across phases.

- Function entities in one phase can be associated with function entities in another phase of the same entity-type, to indicate a self-referencing function activity across phases.

The use of the IRDS to support the Strategic Systems Planning phase is illustrated in this guide. Future publications will address subsequent phases of the system development life cycle. While the IRDS can be used to represent any set of user-defined life cycle phases, a perspective of the system development phases of the life cycle is provided below.

7.2 Early System Development Phases

7.2.1 Strategic Systems Planning

Also known by other names such as Enterprise Analysis, Needs Analysis, and Business Systems Planning, this phase should be conducted **before** other systems development work begins. The **purpose** of Strategic Systems Planning is to establish the context and boundaries for a number of systems development efforts throughout the enterprise, to gain management support and guidance for systems development projects, and to provide the goals, focus, scope, priorities, and high-level requirements for the target systems. The **products** of Strategic Systems Planning are a global Business Model representing global objectives, a global Functional Model, and a high-level, global Information Architecture. Concepts for interrelated information systems are developed as a result of Strategic Systems Planning.

7.2.2 Requirements Definition

The Requirements Definition effort for a particular system, or set of systems, verifies and refines the global Business Model, Functional Model, and Information Architecture that resulted from Strategic Systems Planning, and defines a set of information system requirements. During this phase, subsystems within the system are identified, and technological and performance directions are defined. For the target system and each subsystem, site requirements, communications requirements, data conversion, data processing, and data management requirements are described. The products of Requirements Definition are system-wide functional and data models, subsystem functional and data models, and data flow models between subsystems and systems.

SYSTEM DEVELOPMENT & OPERATIONS LIFE CYCLE

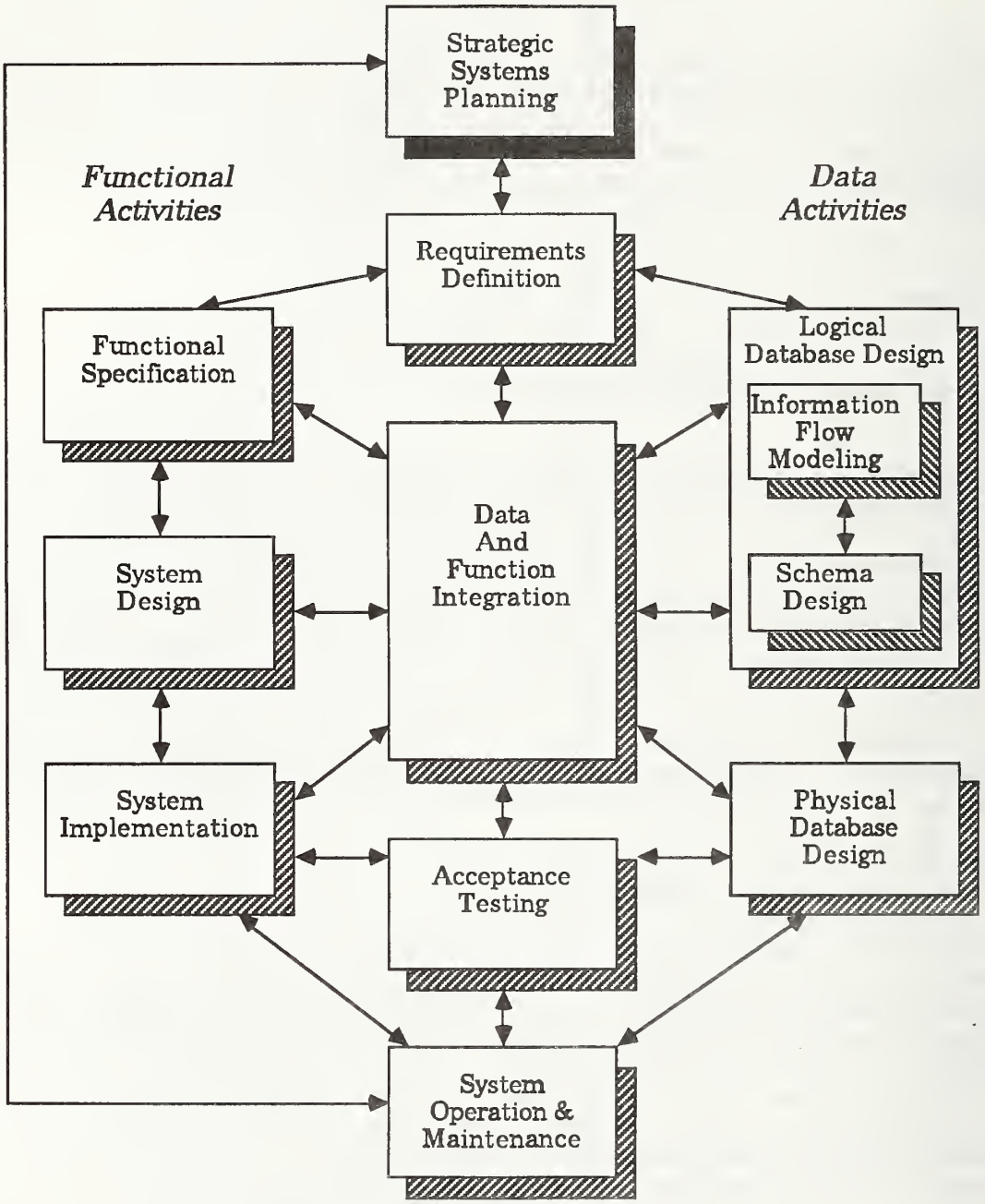


Figure 7

7.3 Intermediate System Development Phases

7.3.1 Functional Specification

The Functional Specification effort further refines the functional models and data flow models from the Requirements Definition phase. The functional models, which had been decomposed down to the subsystem level in Requirements Definition, are now decomposed down through various application levels to represent user-oriented views. The products of Functional Specification are functional decomposition and data flows for each subsystem, functional primitives, data element definitions, and decision-trees.

7.3.2 Logical Database Design

The Logical Database Design effort further refines the data models from Requirements Definition and the data flow models from Functional Specification. The Logical Database Design effort defines both information flow modeling and schema design.

7.3.2.1 Local and Global Information-Flow Modeling

Within this subphase, user-oriented Local Information-flow Models are defined for local entities, and data access workloads are projected for these local entities. As a number of local models are defined, they are consolidated into the Global Information-flow Model to produce global data entities. This consolidation of local entities is an iterative process. System and subsystem boundaries are refined during this subphase. The products of this subphase are many Local Information-flow Models, and the Global Information-flow Model.

7.3.2.2 Conceptual and External Schema Design

Within this subphase, a Conceptual Schema is defined for all information in the system, including identifiers of entities, relationships among entities, the connectivity of relationships, and attributes for entities. All entities are normalized. From the Conceptual Schema, a number of External Schemas are extracted to represent departmental perspectives within the Conceptual Schema. Data access workload information and local constraints are captured for each External Schema. The products of this subphase are the Conceptual Schema and a number of External Schemas.

7.3.3 Data and Function Integration

Data and function integration is a complex process of cross-referencing data structures and process structures within a developing system. Because the integration of data and functions usually takes place iteratively during many life cycle phases, it is rarely considered as a life cycle phase.

Data and function integration is critical to the success of any system development effort. In the desire for rapid system development progress, however, the need for thorough data and function integration can be overlooked. Data and function integration is often poorly performed, resulting in lack of close coordination between the functions and data in the developed system. Since the IRDS provides support for performing data and function integration, this procedure is emphasized in this guide.

7.4 Late System Development Phases

7.4.1 System Design

The Program Design effort partitions the system into high-level modules. Working from the data flow diagrams provided by the Functional Specification and Logical Database Design phases, Structure Charts are defined for each high-level module. The Structure Charts are refined to reduce unnecessary data coupling and promote functional cohesion. Transform analysis is performed to identify and promote the functions performing the central transformation of data. Transaction analysis is performed to factor out common functions into new modules at lower levels, and to combine any shared lower level functions. The product of Program Design is a collection of detailed Structure Charts that show the structure underlying each high-level program module, with the calls and data passing among submodules.

7.4.2 Physical Database Design

The Physical Database Design effort considers data workload factors and the specific DBMS implementation to be used, to produce the detailed Internal Schema for databases used in the target system. The Internal Schema is the basis of the target system's data management function. In defining the Internal Schema, the physical database designers balance the system needs for data storage efficiency, data update facility, and data retrieval performance. The product of Physical Database Design is a detailed Internal Schema that supports database performance optimization.

7.4.3 System Implementation

The Code Implementation effort uses the Structure Charts of Program Design, the data structures of Logical Database Design, and the database implementation structures of Physical Database Design. This phase produces the programs and subroutines that are the basis of the target system's data processing function. Each of the programs and subroutines written are also documented during Code Implementation. Hardware and software configuration management is started during Code Implementation to document the use of particular programs on particular computer hardware to perform particular functions. Subroutines and functions are organized so that they can be reused by multiple programs. The products of Code Implementation are programs, subroutines, functions, code documentation, and hardware and software configuration management.

7.5 Transferring Metadata Across Phases

The command that is used to transfer entities from one life cycle phase to another does **not literally move** the entities within the IRD. Instead, entities are reassigned from one life cycle phase to another, effecting a "logical" not physical transfer. Once you have transferred an entity to the target life cycle phase, the entity is no longer associated with the source life cycle phase.

7.5.1 Transferring Entities

While the command to transfer metadata from one phase to another phase specifies only **entities**, all associated attributes and relationships are also affected. Since all relationships are associated with entities, and all attributes are associated with either entities or relationships, the command to transfer entities to another phase indirectly affects any relationships and attributes associated with those entities. By selecting a number of **entities** associated with a phase, the user will also be able to select the particular **relationships** and **attributes** associated with those entities in that phase.

If a relationship has been defined between an entity in one phase and an entity in another phase, that relationship will span the phases. This phase spanning feature is particularly useful for **cross-referencing** information from phase to phase.

7.5.2 Command to Transfer Entities

The format of the command to transfer entities and related metadata from one phase to another is:

```
Modify entity life-cycle-phase for ENTITY-NAME,
from PHASE-NAME to PHASE-NAME;
```

In this command, the term ENTITY-NAME can be replaced by a list of entity access-names. The use of an entity list permits the user to transfer a large number of entities simultaneously from phase to phase. Examples of the command to transfer an entity from one phase to another are:

```
Modify entity life-cycle-phase for EMPLOYEE,
from REQUIREMENTS-DEFINITION to
LOGICAL-DATABASE-DESIGN-PHASE;

Modify entity life-cycle-phase for ACCOUNTING-SYSTEM,
from SYSTEM-DESIGN-PHASE to
SYSTEM-IMPLEMENTATION-PHASE;

Modify entity life-cycle-phase for PERSONNEL-PAYROLL-DB,
from UNCONTROLLED-LIFE-CYCLE-PHASE to
CONTROLLED-LIFE-CYCLE-PHASE;
```

7.5.3 Limitations to Metadata Transfer Between Phases

IRD metadata can be transferred only between certain life cycle phases. As stated previously in this guide, the IRDS provides a predefined UNCONTROLLED-LIFE-CYCLE-PHASE that corresponds to the system development phase, a predefined CONTROLLED-LIFE-CYCLE-PHASE that corresponds to the system operation phase, and a predefined ARCHIVED-LIFE-CYCLE-PHASE that corresponds to the maintenance of historical records. Users can define additional Uncontrolled life cycle phase partitions for use during system development.

IRD users can transfer metadata either: (1) from the UNCONTROLLED-LIFE-CYCLE-PHASE to one of the user-defined Uncontrolled life cycle phases; (2) from a user-defined Uncontrolled life cycle phase to the UNCONTROLLED-LIFE-CYCLE-PHASE; (3) from the UNCONTROLLED-LIFE-CYCLE-PHASE to the CONTROLLED-LIFE-CYCLE-PHASE; (4) from the CONTROLLED-LIFE-CYCLE-PHASE to the ARCHIVED-LIFE-CYCLE-PHASE; or (5) from the ARCHIVED-LIFE-CYCLE-PHASE to the CONTROLLED-LIFE-CYCLE-PHASE. Users **cannot** transfer metadata from the UNCONTROLLED-LIFE-CYCLE-PHASE, or other user-defined Uncontrolled life cycle phases, to the ARCHIVED-LIFE-CYCLE-PHASE.

8.0 IRDS Support for Strategic Systems Planning

The IRDS can support any system development life cycle phases that the user wants to define, such as those shown in Figure 7. When an enterprise follows the policies of Information Resource Management, the first phase in system development is **Strategic Systems Planning.** Because this first phase encompasses all systems and organizational planning within an enterprise, or within a major component of an enterprise, the scope of Strategic Systems Planning is greater than that of any one system. Strategic Systems Planning results in a high-level, enterprise-wide Information Architecture that provides a foundation for integrated system development projects throughout the enterprise.

The purpose of Strategic Systems Planning is to describe a high-level **Information Architecture** for the entire enterprise that provides a global perspective of the following:

- Critical success factors, or primary objectives, for the success of the enterprise.
- Functions and organizational structures within the enterprise.
- Processes and data needed within the enterprise.
- Plans for systems development and other projects to reach those objectives.
- Management objectives to achieve the enterprise's critical success factors, and management support for projects designed to meet these objectives.

Although only high-level information is defined during Strategic Systems Planning, a considerable amount of information can be collected. The support of a data dictionary system such as the IRDS is important not only to the success of this effort, but also to insure that the information derived from this effort is easily accessible for reference during the subsequent system development phases.

8.1 Strategic Systems Planning Phase Description

A series of procedures should be conducted during Strategic Systems Planning to describe an Information Architecture [FONG86], [MART82]. This chapter describes a number of these procedures for Strategic Systems Planning.

8.1.1 Analysis of Global Business Objectives

The Global Business Objectives defined during this step should include: (1) critical success factors for the progress of the enterprise, and (2) management objectives to fulfill those success factors. **Critical success factors** are goals defined by high-level management as being critical to the success of the enterprise. Usually only a small number of critical success factors, such as three to six, are identified. Once defined, these critical success factors provide the goals for subsequent management objectives and provide the basis for enterprise analysis.

Management objectives provide measurable goals to be fulfilled at every level of the enterprise. The procedures for management-by-objective should also be applied to information systems, to insure that information systems development and modification projects support the critical success factors defined for the enterprise.

8.1.2 Definition of a Global Business Model

The Global Business Model defined during this step includes a definition of the existing organizational structure, a functional decomposition of all the major procedures performed within the enterprise, a mapping of the functional decomposition against the existing organizational structures, and a proposal of alternatives for a more efficient organizational structure.

To define the **existing organizational structure**, the organizational components of the enterprise are assigned to a hierarchical structure similar to that of an organizational chart. The representation of the existing organizational structure can be modified later in the phase, if necessary, to show a new organizational structure.

The **functional decomposition** of functions performed within the enterprise includes the definition of: (1) the **major functional areas** of the enterprise, (2) the **major processes** performed within those functional areas, and (3) the **major activities** supporting those processes. While most **functional areas** and some **processes** can be defined during Strategic Systems Planning, **activities** are usually defined during subsequent system development phases.

The results of the functional decomposition are represented hierarchically in an IRD so that the functional areas "contain" a number of processes, and the processes "contain" a number of activities.

In **mapping the functional decomposition against the organizational structure**, the functions and processes of the functional decomposition are cross-referenced to the organizational structure. Analysis of this mapping provides information for a possible **reorganization of the enterprise.**

When a large number of organizational components are performing the same function or process, redundant organizational components might be put to better use in other functional areas. When a very small number of organizational components are performing an important function or process, this area is weak and should be reinforced. A number of **alternative reorganization plans** can be proposed for management to consider.

8.1.3 Definition of a Global Data Model

The Global Data Model described during this step includes description of the major data used within the enterprise as the: (1) **subject data areas**, or subject databases, (2) **data sets**, and (3) **data entities. Subject data areas** are the primary data categories within the enterprise. These high-level subject data areas must be defined independently of the organizational structures that use them. The term "subject data area" is used in this guide rather than "subject database" because this concept does not directly correspond to a physical database. When implemented, the information represented by a subject data area may be stored in one or more databases or files.

Once a number of subject data areas have been defined, such as ten to thirty, then a number of major **data sets** should be described within each area. A data set is a generalized group of information that can be structured within a subject data area. Data sets can be decomposed to indicate a few high-level data **entities** that represent the most important concepts of a data set. A limited number of **data entities** may be defined during Strategic Systems Planning, while the majority of data entities can be defined later during Requirements Definition. During **later life cycle phases**, each data entity will be decomposed into one or more **data elements.**

Data elements are the lowest, most basic level of the data structure. Because only high-level data description is pursued throughout Strategic Systems Planning, data elements are not defined during this phase.

The results of this data area modeling are represented hierarchically in an IRD so that the subject data areas "contain" a number of data sets, and the data sets "contain" a number of data entities.

In **mapping the data model against the organizational structure**, the subject data areas and data sets are cross-referenced to planned organizational structures. Analysis of this mapping provides information for a possible **reorganization of databases or data files** of the enterprise.

When several organizational components are using the same subject data areas, databases or files should be reorganized so these organizational components can share this data. A number of **alternative database reorganization plans** can be proposed for management to consider.

8.1.4 Data and Function Cross-Referencing

Once the data and function modeling has been completed, then each **functional area** can be mapped to the **subject data areas** needed to perform that function. Additionally each **process** can then be mapped to the **data sets** needed to support that process. If activities and data entities have been defined, each **activity** can be mapped to the **data entities** needed to perform that activity. This data and function cross-referencing indicates what types of data are used in performing the primary functions of the enterprise, and shows the relative importance of the primary data areas.

The subject data areas that are associated with many functions should receive a higher priority than the data areas associated with few or no major functions. If a particular data set or data entity has many associations with processes or activities, consider whether that data set or data entity should be promoted to the next higher level.

8.1.5 Assessment of Enterprise Directions

The highest level of management must participate in setting the directions for the evolution of the enterprise. Many types of directions can be defined during this step. Some enterprise directions to be considered are: (1) **technological directions** for the development or use of new technology; (2) **organizational directions** for redefined goals or organizational restructuring; (3) **skill and training directions** for the skill categories to be emphasized in hiring and training programs; and (4) **project directions** for the near-term and long-term projects needed to meet management objectives and fulfill the critical success factors of the enterprise.

The Strategic Systems Planning group will usually propose a number of projects to be performed throughout the enterprise. These proposed projects will often involve information systems development or modification.

Information system development should be influenced by the directions defined for the enterprise. The **technological directions** of the enterprise will provide system designers with information about management's level of interest in new technology. The **organizational directions** show the goals of the enterprise that system development or redesign projects are designed to fulfill. The **training and skill directions** indicate management's emphasis on certain skill categories that system development efforts may either use directly or indirectly enhance. The **project directions** provide project scheduling information, and show how system development projects fit into the larger scheme of management objectives for the enterprise.

Successful information system development projects fit into the defined enterprise directions and receive direct management support in meeting defined management objectives.

8.2 Strategic Systems Planning Application

The example Strategic Systems Planning application includes the definition of critical success factors and organizational structures, the decomposition of functional areas and processes, the decomposition of subject data areas and data sets, and the cross-referencing of functional area, subject data area, and organizational information. The examples shown do not fully illustrate all the steps of this phase.

8.2.1 Critical Success Factors

In Figure 8, a number of **critical success factors** for the success of the enterprise have been defined by the high-level management of the XYZ Corporation, a manufacturing enterprise.

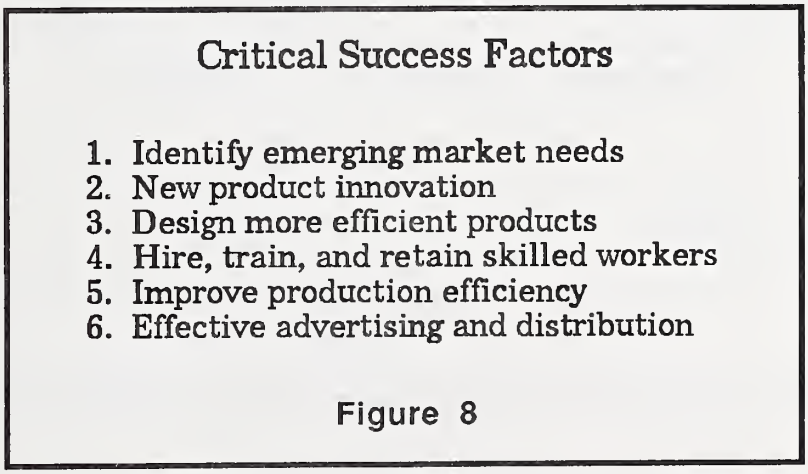


Figure 8

8.2.2 Organizational Structures

Figure 9 illustrates the existing high-level organizational structures of the XYZ Corporation. The hierarchical structure is typical of most organizational charts.

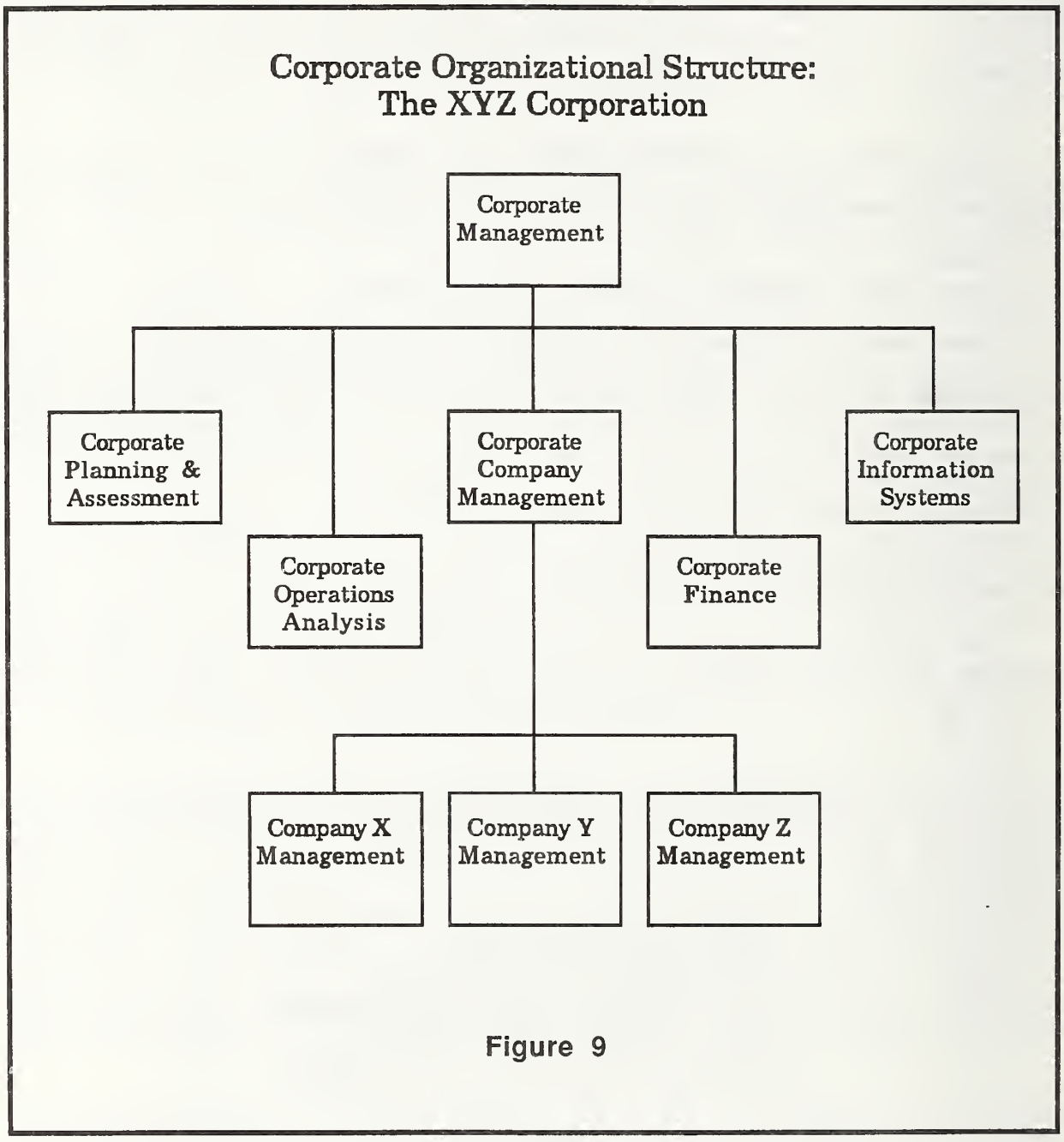


Figure 9

Figure 10 illustrates some of the existing and planned organizational structures of Company X, a subsidiary of the XYZ Corporation. The high-level organizational structures of the company are shown along with the mid-level structures of the Accounting Division.

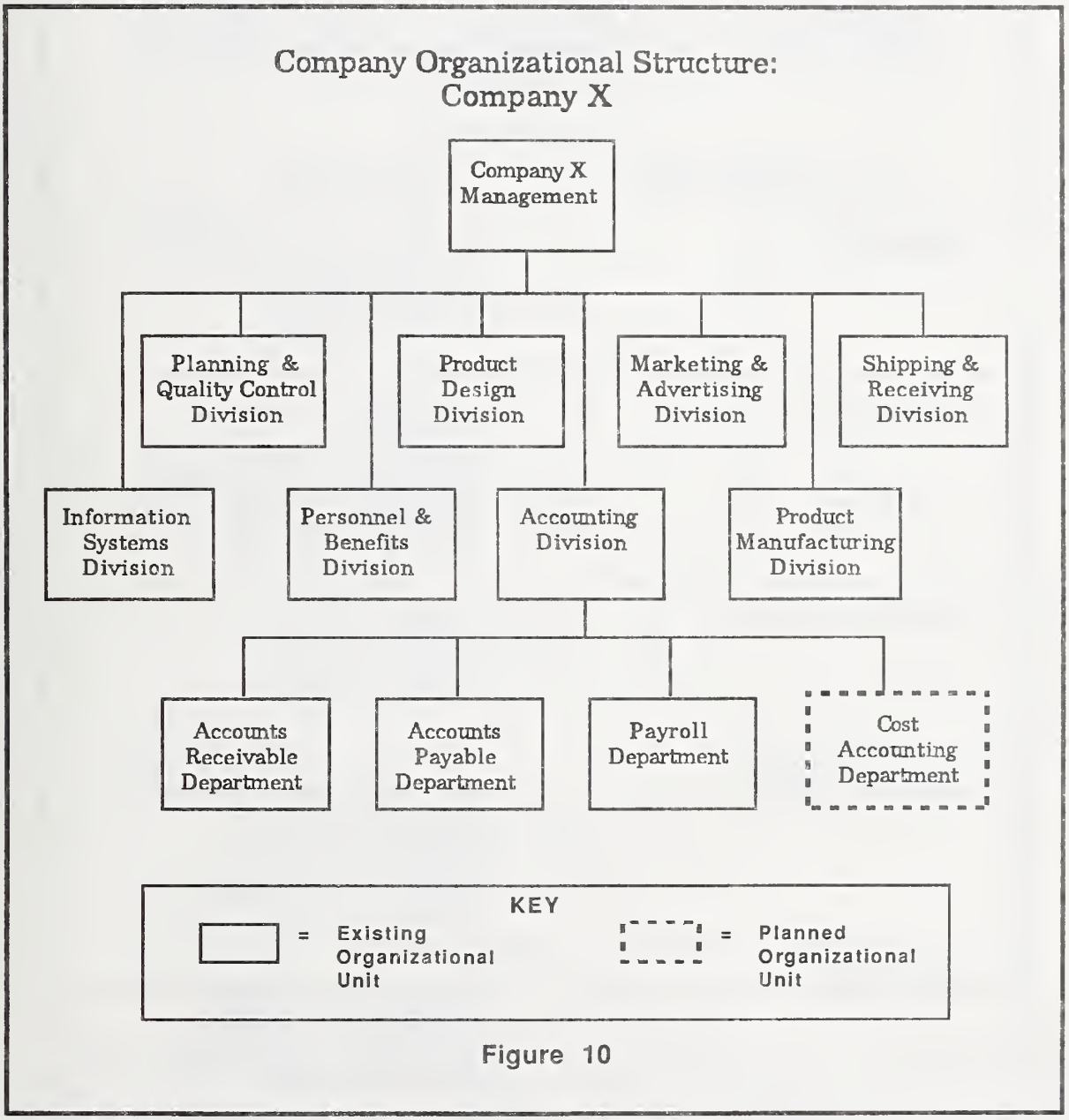


Figure 10

8.2.3 Decomposition and Cross-Referencing

Figure 11 illustrates a small part of a high-level, enterprise-wide functional decomposition that defines functional areas and processes. The functional areas defined do not necessarily correspond to existing organizational components. An organizational unit may support part of a functional area, or one or more functional areas.

High-Level Functional Decomposition for Company X

FUNCTIONAL AREAS	PROCESSES
ACCOUNTING	ACCOUNTS RECEIVABLE ACCOUNTS PAYABLE EMPLOYEE PAYROLL COST ACCOUNTING TAX ACCOUNTING
PERSONNEL	PERSONNEL PLANNING RECRUITING EMPLOYEE TRAINING EMPLOYEE COMPENSATION EMPLOYEE BENEFITS EMPLOYEE INCENTIVES
PRODUCT PLANNING	PRODUCT DESIGN PRODUCT ANALYSIS PRODUCT REDESIGN PRODUCT OPTIONS
PRODUCT MARKETING	PRODUCT MARKET ANALYSIS SALES FORECASTING PRODUCT PRICING PRODUCT PACKAGING PRODUCT ADVERTISING PRODUCT DISTRIBUTION
PRODUCTION	PRODUCTION SCHEDULING PROCEDURE PLANNING MATERIALS INVENTORY EQUIPMENT SCHEDULING QUOTA REPORTING

Figure 11

Figure 12 illustrates the cross-referencing of critical success factors to functional areas. Each SUCCESS-FACTOR is shown in relation to each FUNCTIONAL-AREA that it influences.

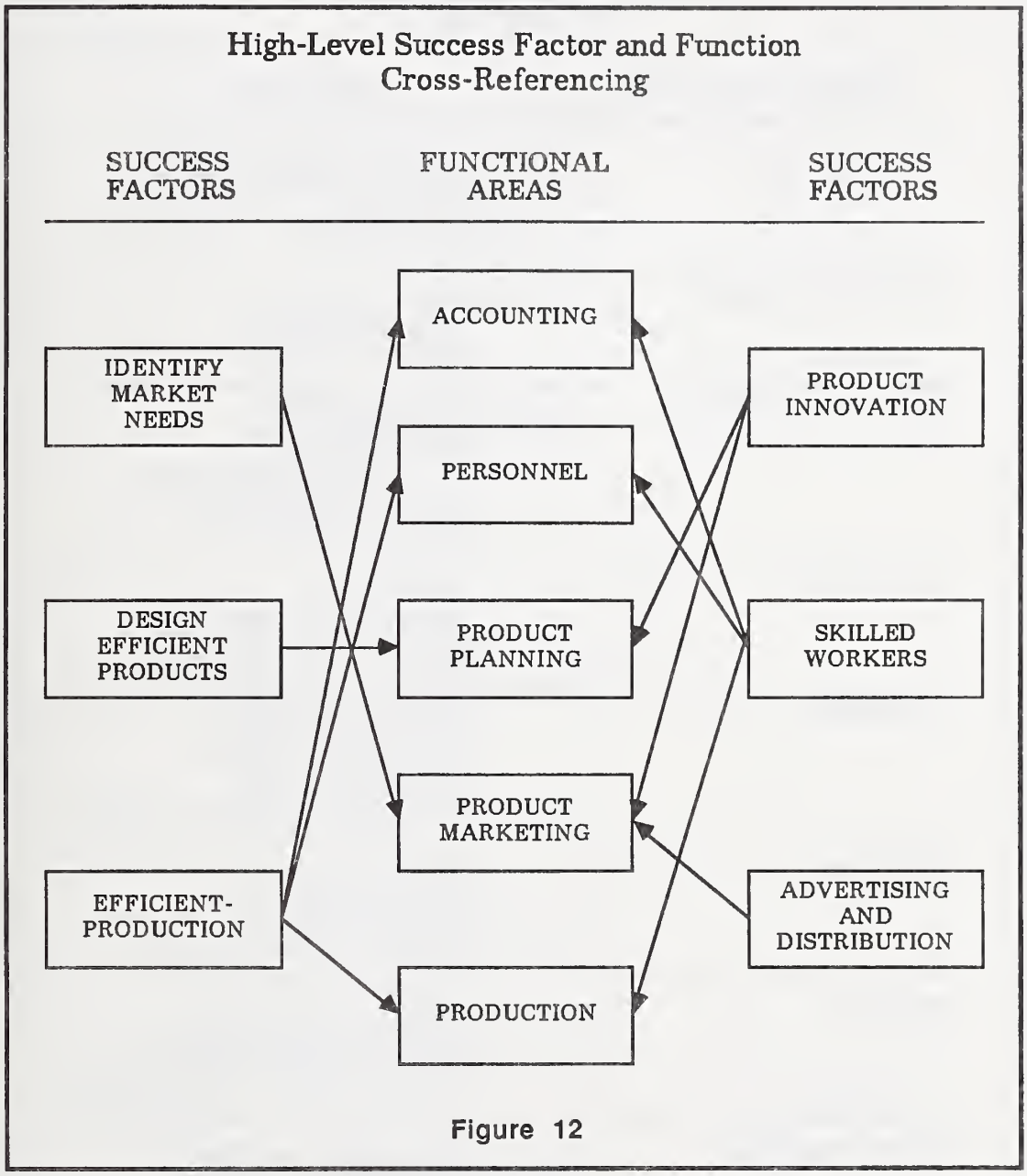


Figure 12

Part of a high-level data decomposition is illustrated in Figure 13 with a number of subject data areas and data sets. The conceptual groupings represented by these subject data areas are **not** intended to correspond directly to existing or planned databases or files.

High-Level Data Decomposition for Company X

SUBJECT DATA AREAS	DATA SETS
EMPLOYEE	PERSONAL INFORMATION POSITION DESCRIPTION EDUCATION PERFORMANCE
EMPLOYEE PAYMENTS	PAY CATEGORY AWARD CATEGORY BONUS CATEGORY
EMPLOYEE BENEFITS	LEAVE CATEGORY RETIREMENT PAY RETIREMENT SAVINGS CO SAVINGS CONTRIBUTION CO INSURANCE CONTRIBUTION CO TAX CONTRIBUTION
DEPARTMENT	PROJECT DEPARTMENT EMPLOYEES DEPARTMENT CONTRACTS DEPARTMENT PRODUCTS
CUSTOMER	CUSTOMER LOCATION CUSTOMER ACCOUNT CUSTOMER CONTRACTS CUSTOMER ORDERS CUSTOMER PAYMENTS RECEIVED
SUPPLIER	SUPPLIER LOCATION SUPPLIER CONTRACTS SUPPLIER ORDERS SUPPLIER PAYMENTS SENT
PRODUCT	PRODUCT DESCRIPTION PRODUCT SPECIFICATIONS PRODUCT PRODUCTION SCHEDULES PRODUCT MAINTENANCE MANUAL
PRODUCT MARKET	PRODUCT MARKET SHARE PRODUCT COMPETITORS COMPETITOR PRODUCTS PRODUCT DISTRIBUTORS PRODUCT PRICE PRODUCT SALES PERFORMANCE

Figure 13

Figure 14 illustrates data and function cross-referencing through the association of a few functional areas with a small number of subject data areas. The functional areas are shown in relation to the subject data areas needed to perform those functions.

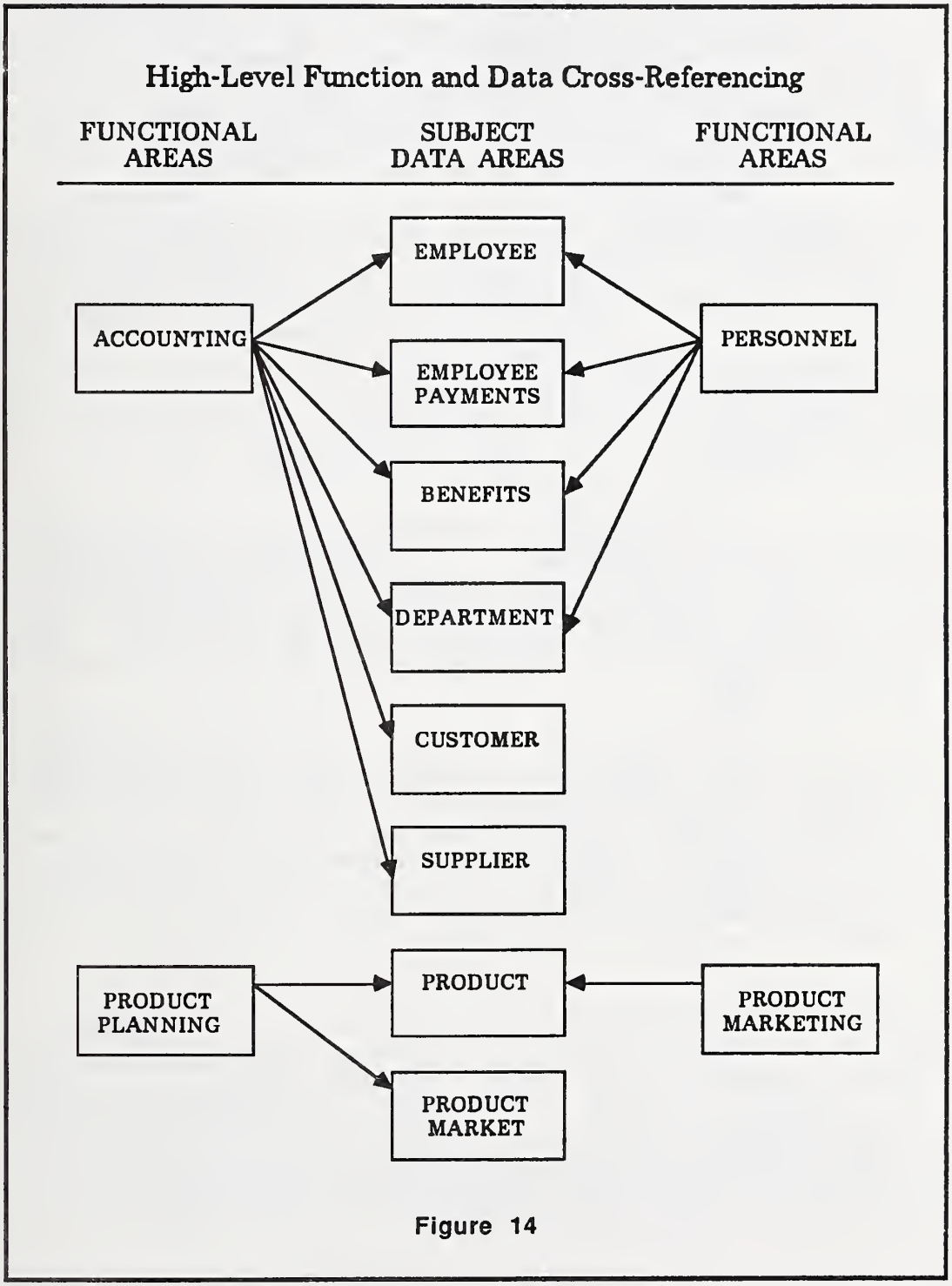


Figure 14

8.3 Strategic Systems Planning Entity-Relationship Models

The corporate and company organizational structures of Figures 9 and 10 translate into the Entity-Relationship-Attribute model of Figure 15. The entity **COMPANY-X** is shown with an attribute **CHICAGO** of attribute-type **location**. **COMPANY-X** is also shown in an association of **IS-PART-OF** with entity **CORPORATION-XYZ**. In turn, the entity **COMPANY-X** has a association of **CONTAINS** with entity **ACCOUNTING-DIVISION**. This entity also has an association of **CONTAINS** with the entity **PAYROLL-DEPT**. The entity **EMPLOYEE-PAYROLL** of entity-type **process** is shown in an association of **IS-PERFORMED-BY** with the entity **PAYROLL-DEPT**.

In Figure 16, the critical success factors of Figure 8 translate easily into entities, which can be defined for an IRD as of entity-type **success-factor**. The functional areas and processes of Figure 11 become entity-type **functional-area** and entity-type **process**. Similarly, the subject data areas and data sets of Figure 12 become entity-type **data-area** and entity-type **data-set**. The relationships between entities of entity-type **success-factor** and entities of entity-type **functional-area** can be defined in terms of relationship-class-type **INFLUENCES**, which describes an indirect, semi-causal association.

Similarly, relationships between entities of entity-type **functional-area** and entities of entity-type **process** can be defined in terms of the relationship-class-type **CONTAINS**, which describes a direct, hierarchical association. The relationships between entities of entity-type **data-area** and entities of entity-type **data-set** can also be defined in terms of the relationship-class-type **CONTAINS**.

The cross-referencing between entities of entity-type **functional-area** and entity-type **data-area**, shown in Figure 14, can be defined in terms of the relationship-class-type **refers-to**, which describes an indirect referential association. Since we do not know yet whether the **functional-area** entity-type "uses" or "creates" the data in the **data-area** entity-type, the general term "refers to" is used. Later in the life cycle, a more specific relationship-class-type can be defined to support this association.

To show which success factor **influences** a functional area, the term **"influences"** becomes the central term in the relationship between entities of types **success-factor** and **functional-area**. In this figure, the entity **EFFICIENT-PRODUCTION** of entity-type **success-factor** has an association of **INFLUENCES** with entity **PERSONNEL** of the entity-type **functional-area**. The relationship-class-type **INFLUENCES** provides the central term for the relationship-type **SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA**.

Entity-Relationship-Attribute Model for Organizational Structure

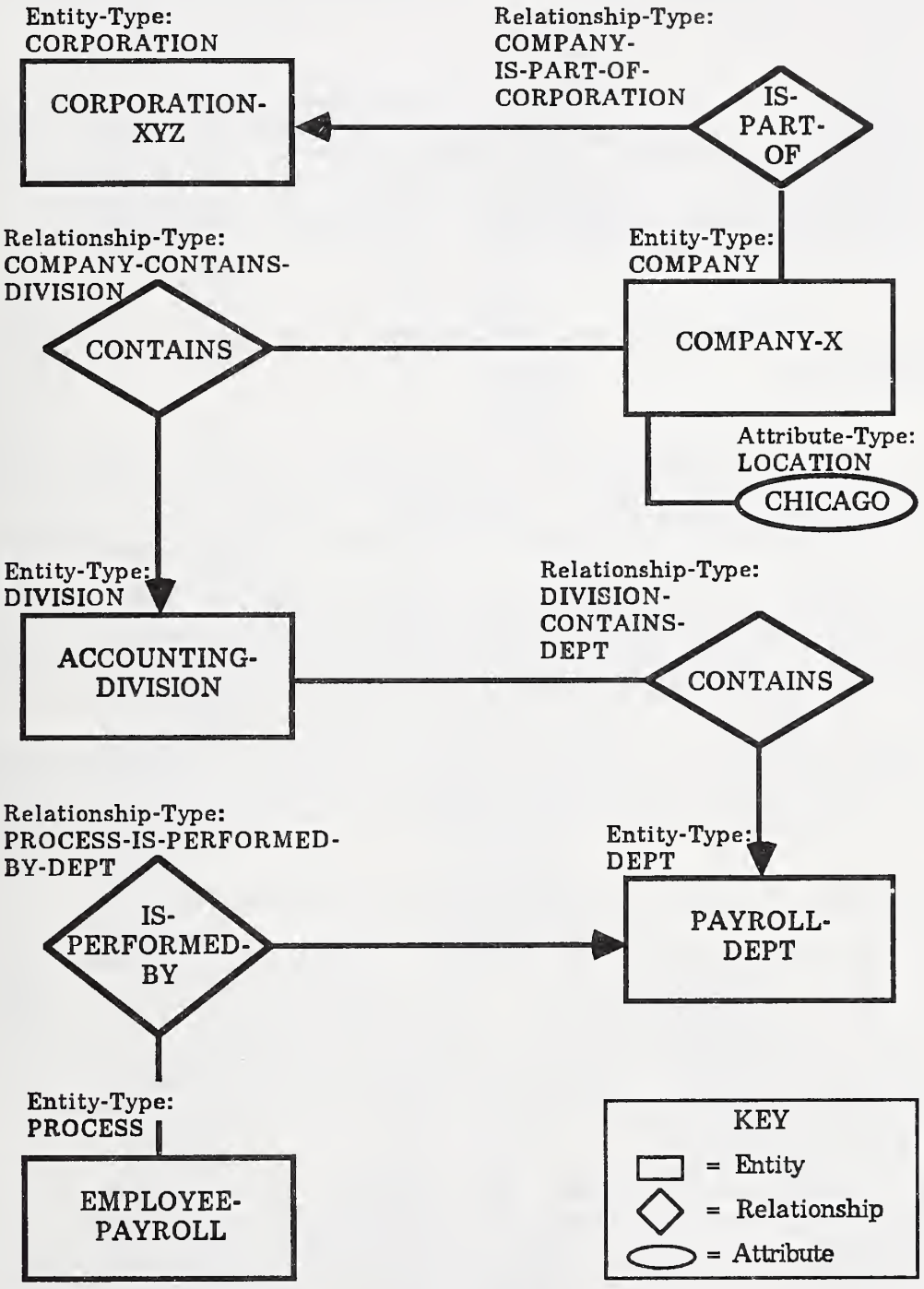


Figure 15

The full relationship between the entities EFFICIENT-PRODUCTION and PERSONNEL in Figure 16 can be represented as:

EFFICIENT-PRODUCTION SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA PERSONNEL

Similarly, Figure 16 shows **PERSONNEL** of the entity-type **functional-area** in an association of **CONTAINS** with **EMPLOYEE-INCENTIVES** of the entity-type **process**. The relationship-class-type **CONTAINS** provides the central term for the relationship-type **FUNCTIONAL-AREA-CONTAINS-PROCESS**. The full relationship between FUNCTIONAL-AREA and PROCESS is:

PERSONNEL FUNCTIONAL-AREA-CONTAINS-PROCESS EMPLOYEE-INCENTIVES.

The entity **PERSONNEL** is also shown in an association of **REFERS-TO** the entity **EMPLOYEE-PAYMENTS** of entity-type **data-area**. The relationship-class-type **REFERS-TO** provides the central term for the relationship-type **FUNCTIONAL-AREA-REFERS-TO-DATA-AREA**.

Similarly, the entity **EMPLOYEE-PAYMENTS** is shown in an association of **CONTAINS** with the entity **PAY-CATEGORY** of entity-type **data-set**. The relationship-class-type **CONTAINS** is the basis for the relationship-type **DATA-AREA-CONTAINS-DATA-SET**.

Figures 15 and 16 provide the Entity-Relationship-Attribute model that can be used in representing Strategic Systems Planning with the IRDS in an Information Resource Dictionary, or IRD.

8.4 Strategic Systems Planning Schema Definition

This section shows an example **schema** definition that will support the sample metadata for Strategic Systems Planning. Although not described here, attribute-types can be added to entity-types and relationship-types, as described in Chapter 3. Since the relationship-class-type **CONTAINS** and the attribute-type **LOCATION** are already predefined in the Basic Functional Schema, they are not added here.

Each **entity-type name** and **entity name** must be **unique** throughout an IRD, regardless of the phase in which it is used. Users who want to use the same entity-type name or entity name in multiple phases can add a phase-related **assigned access name suffix** (such as "-P" for Planning) to differentiate each **entity-type name**, and a **variation-name suffix** (such as "P" for Planning) to differentiate each **entity name** used in a different phase.

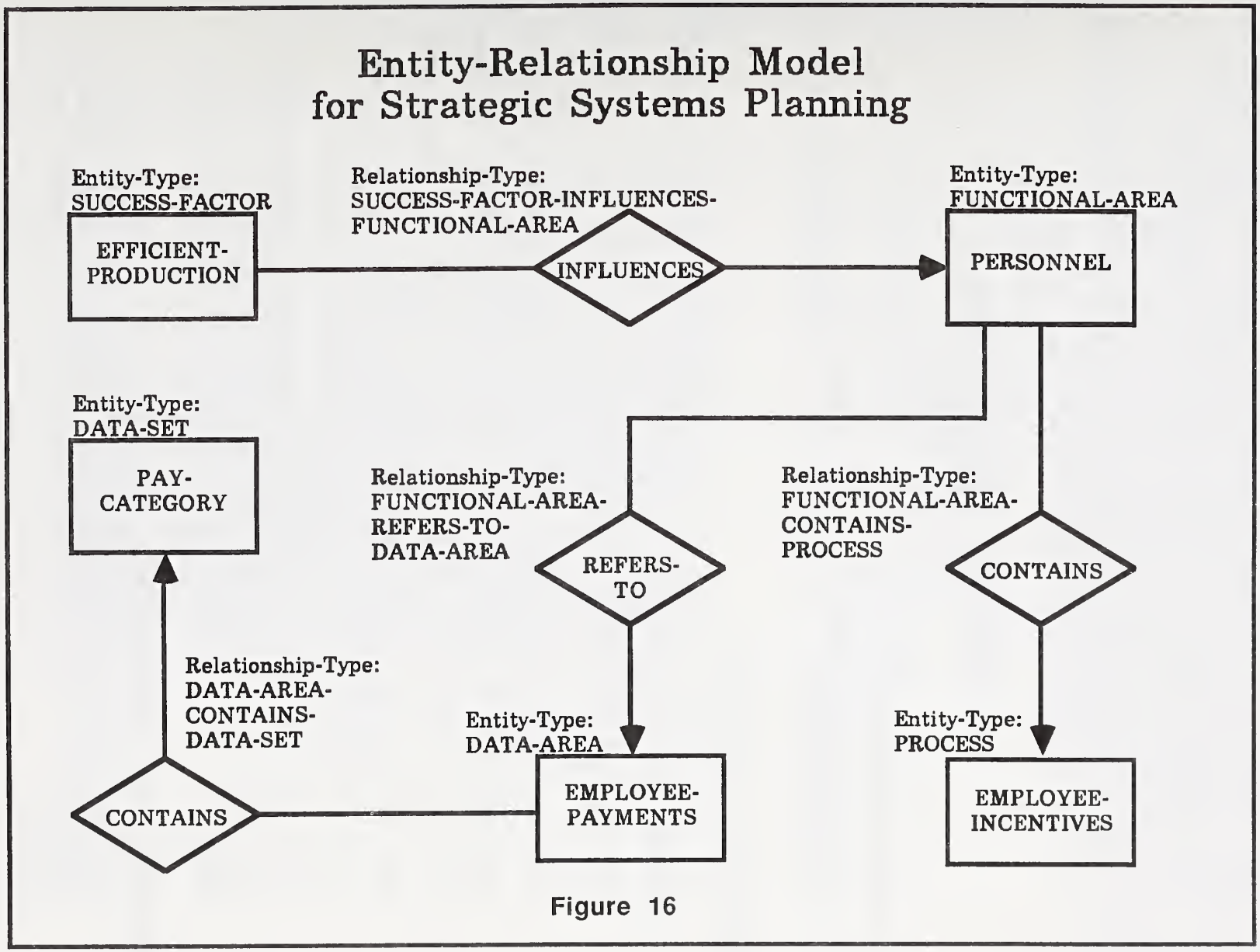

Entity-Relationship Model
for Strategic Systems Planning
Entity-Type:
SUCCESS-FACTOR
EFFICIENT-
PRODUCTION
Relationship-Type:
SUCCESS-FACTOR-INFLUENCES-
FUNCTIONAL-AREA
INFLUENCES
Entity-Type:
FUNCTIONAL-AREA
PERSONNEL
Entity-Type:
DATA-SET
PAY-
CATEGORY
Relationship-Type:
FUNCTIONAL-AREA-
REFERS-TO-
DATA-AREA
REFERS-
TO
Relationship-Type:
FUNCTIONAL-AREA-
CONTAINS-
PROCESS
CONTAINS
Relationship-Type:
DATA-AREA-
CONTAINS-
DATA-SET
Entity-Type:
DATA-AREA
Entity-Type:
PROCESS
CONTAINS
EMPLOYEE-
PAYMENTS
EMPLOYEE-
INCENTIVES

Figure 16

The addition the suffix "-P" (for Planning) at the end of each meta-entity assigned access name representing an entity-type indicates that this entity-type is defined in the **Strategic Systems Planning** phase, and gives the name a unique definition for this phase. Since the meta-entity of **location** has already been predefined as an **attribute-type** in the Basic Functional Schema, it is not added here, but is modified for this application. The following IRDS commands define the **schema** for the life cycle phase STRATEGIC-SYSTEMS-PLANNING.

STRATEGIC SYSTEMS PLANNING
SCHEMA DEFINITION

Life Cycle Phase Definition:

Add meta-entity **STRATEGIC-SYSTEMS-PLANNING** meta-entity-type = **IRD-Partition**;

Entity-Type Definition:

Add meta-entity **SUCCESS-FACTOR-P** meta-entity-type = **entity-type**;

Add meta-entity **FUNCTIONAL-AREA-P** meta-entity-type = **entity-type**;

Add meta-entity **PROCESS-P** meta-entity-type = **entity-type**;

Add meta-entity **DATA-AREA-P** meta-entity-type = **entity-type**;

Add meta-entity **DATA-SET-P** meta-entity-type = **entity-type**;

Add meta-entity **DIVISION-P** meta-entity-type = **entity-type**;

Add meta-entity **DEPT-P** meta-entity-type = **entity-type**;

Add meta-entity **COMPANY-P** meta-entity-type = **entity-type**;

Add meta-entity **CORPORATION-P** meta-entity-type = **entity-type**;

Attribute-Type Definition:

Modify meta-entity **LOCATION** with **purpose** = "Indicates the location of a company or other corporate organizational unit";

Entity-Type Associated with Attribute-Type:

```
Add meta-relationship COMPANY-P contains LOCATION;
```

Relationship-Class-Type Definition:

```
Add meta-entity INFLUENCES meta-entity-type =
relationship-class-type;

Add meta-entity REFERS-TO meta-entity-type =
relationship-class-type;

Add meta-entity IS-PART-OF meta-entity-type =
relationship-class-type;

Add meta-entity IS-PERFORMED-BY meta-entity-type = relation-
ship-class-type;
```

Relationship-Type Definition:

```
Add meta-entity SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
meta-entity-type = relationship-type;

Add meta-entity FUNCTIONAL-AREA-CONTAINS-PROCESS
meta-entity-type = relationship-type;

Add meta-entity FUNCTIONAL-AREA-REFERS-TO-DATA-AREA
meta-entity-type = relationship-type;

Add meta-entity DATA-AREA-CONTAINS-DATA-SET
meta-entity-type = relationship-type;

Add meta-entity COMPANY-IS-PART-OF-CORPORATION
meta-entity-type = relationship-type;

Add meta-entity COMPANY-CONTAINS-DIVISION
meta-entity-type = relationship-type;

Add meta-entity DIVISION-CONTAINS-DEPT
meta-entity-type =  relationship-type;

Add meta-entity PROCESS-IS-PERFORMED-BY-DEPT
meta-entity-type = relationship-type;
```

Relationship-Type Associated with Relationship-Class-Type:

```
Add meta-relationship SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-
AREA member-of INFLUENCES;

Add meta-relationship FUNCTIONAL-AREA-CONTAINS-PROCESS
member-of CONTAINS;

Add meta-relationship FUNCTIONAL-AREA-REFERS-TO-DATA-AREA
member-of REFERS-TO;

Add meta-relationship DATA-AREA-CONTAINS-DATA-SET
member-of CONTAINS;

Add meta-relationship COMPANY-IS-PART-OF-CORPORATION
member-of IS-PART-OF;

Add meta-relationship COMPANY-CONTAINS-DIVISION
member-of CONTAINS;

Add meta-relationship DIVISION-CONTAINS-DEPT
member-of CONTAINS;

Add meta-relationship PROCESS-IS-PERFORMED-BY-DEPT
member-of IS-PERFORMED-BY;
```

Relationship-Type Positional Definition:

```
Add meta-relationship SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-
AREA connects SUCCESS-FACTOR-P position = 1;

Add meta-relationship SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-
AREA  connects FUNCTIONAL-AREA-P position = 2;


Add meta-relationship FUNCTIONAL-AREA-CONTAINS-PROCESS
connects  FUNCTIONAL-AREA-P position = 1;

Add meta-relationship FUNCTIONAL-AREA-CONTAINS-PROCESS
connects  PROCESS-P position = 2;


Add meta-relationship FUNCTIONAL-AREA-REFERS-TO-DATA-AREA
connects FUNCTIONAL-AREA-P position = 1;

Add meta-relationship FUNCTIONAL-AREA-REFERS-TO-DATA-AREA
connects DATA-AREA-P position = 2;
```

```
Add meta-relationship DATA-AREA-CONTAINS-DATA-SET
connects DATA-AREA-P position = 1;

Add meta-relationship DATA-AREA-CONTAINS-DATA-SET
connects  DATA-SET-P position = 2;


Add meta-relationship COMPANY-IS-PART-OF-CORPORATION
connects  COMPANY-P position = 1;

Add meta-relationship COMPANY-IS-PART-OF-CORPORATION
connects CORPORATION-P position = 2;


Add meta-relationship COMPANY-CONTAINS-DIVISION
connects  COMPANY-P position = 1;

Add meta-relationship COMPANY-CONTAINS-DIVISION
connects DIVISION-P position = 2;


Add meta-relationship DIVISION-CONTAINS-DEPT
connects  DIVISION-P position = 1;

Add meta-relationship DIVISION-CONTAINS-DEPT
connects DEPT-P position = 2;


Add meta-relationship PROCESS-IS-PERFORMED-BY-DEPT
connects PROCESS-P position = 1;

Add meta-relationship PROCESS-IS-PERFORMED-BY-DEPT
connects DEPT-P position = 2;
```

8.5 Strategic Systems Planning Metadata Definition

This section shows how metadata about Strategic Systems Planning can be represented in an IRD. Because every entity name must be unique throughout an IRD, and since some of the same entity names will be used during both the Strategic Systems Planning and the Requirements Definition phases, a variation-name suffix is added to each entity name.

The variation-name "P" added in parenthesis at the end of each meta-entity name indicates that this entity is defined in the Strategic Systems Planning phase, and gives the term a unique variation for this phase. The following IRDS commands define views and some metadata for the life cycle phase STRATEGIC-SYSTEMS-PLANNING. Since view names cannot have version identifiers, the phase name is included in the view name.

STRATEGIC SYSTEMS PLANNING
METADATA DEFINITION

View Definition within a Life Cycle Phase:

```
Add entity TOTAL-VIEW-PLANNING entity-type = IRD-View with
IRD-Partition-Name = STRATEGIC-SYSTEMS-PLANNING;
```

View Access Permission Command:

```
Add relationship LAW user-has-IRD-view TOTAL-VIEW-PLANNING;
```

User's Effective-View Definition:

```
Set IRD view = TOTAL-VIEW-PLANNING;
```

Entity Definition for Entity-Type CORPORATION:

```
Add entity CORPORATION-XYZ(P) entity-type = CORPORATION-P;
```

Entity Definition for Entity-Type COMPANY:

```
Add entity COMPANY-X(P) entity-type = COMPANY-P with
LOCATION = "CHICAGO";

Add entity COMPANY-Y(P) entity-type = COMPANY-P with
LOCATION = "ST. LOUIS";

Add entity COMPANY-Z(P) entity-type = COMPANY-P with
LOCATION = "CINCINNATI";
```

Entity Definition for Entity-Type DIVISION:

```
Add entity PERSONNEL-BENEFITS-DIVISION(P) entity-type =
DIVISION-P;

Add entity ACCOUNTING-DIVISION(P) entity-type =
DIVISION-P;

Add entity PRODUCT-DESIGN-DIVISION(P) entity-type =
DIVISION-P;
```

```
Add entity MARKETING-ADVERTISING-DIVISION(P) entity-type =
DIVISION-P;
```

Entity Definition for Entity-Type DEPT:

```
Add entity ACCOUNTS-RECEIVABLE-DEPT(P) entity-type =  DEPT-
P;

Add entity ACCOUNTS-PAYABLE-DEPT(P) entity-type =
DEPT-P;

Add entity PAYROLL-DEPT(P) entity-type =
DEPT-P;

Add entity COST-ACCOUNTING-DEPT(P) entity-type =
DEPT-P;
```

Entity Definition for Entity-Type SUCCESS-FACTOR:

```
Add entity IDENTIFY-MARKET-NEEDS(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
IDENTIFY-EMERGING-MARKET-NEEDS;

Add entity PRODUCT-INNOVATION(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
NEW-PRODUCT-INNOVATION;

Add entity DESIGN-EFFICIENT-PRODUCTS(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
DESIGN-MORE-EFFICIENT-PRODUCTS;

Add entity SKILLED-WORKERS(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
HIRING-TRAINING-AND-RETENTION-OF-SKILLED-WORKERS;

Add entity EFFICIENT-PRODUCTION(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
MORE-EFFICIENT-PRODUCTION;

Add entity ADVERTISING-AND-DISTRIBUTION(P) entity-type =
SUCCESS-FACTOR-P entity descriptive-name =
MORE-EFFECTIVE-ADVERTISING-AND-DISTRIBUTION;
```

Entity Definition for Entity-Type FUNCTIONAL-AREA:

```
Add entity ACCOUNTING(P) entity-type = FUNCTIONAL-AREA-P;

Add entity PERSONNEL(P) entity-type = FUNCTIONAL-AREA-P;

Add entity PRODUCT-PLANNING(P) entity-type =
FUNCTIONAL-AREA-P;

Add entity PRODUCT-MARKETING(P) entity-type =
FUNCTIONAL-AREA-P;

Add entity PRODUCTION(P) entity-type = FUNCTIONAL-AREA-P;
```

Entity Definition for Entity-Type PROCESS:

```
Add entity ACCOUNTS-RECEIVABLE(P) entity-type = PROCESS-P;

Add entity ACCOUNTS-PAYABLE(P) entity-type = PROCESS-P;

Add entity EMPLOYEE-PAYROLL(P) entity-type = PROCESS-P;

Add entity COST-ACCOUNTING(P) entity-type = PROCESS-P;

Add entity PERSONNEL-PLANNING(P) entity-type = PROCESS-P;

Add entity RECRUITING(P) entity-type = PROCESS-P;

Add entity EMPLOYEE-TRAINING(P) entity-type = PROCESS-P;

Add entity EMPLOYEE-COMPENSATION(P) entity-type = PROCESS-P;

Add entity EMPLOYEE-BENEFITS(P) entity-type = PROCESS-P;

Add entity EMPLOYEE-INCENTIVES(P) entity-type = PROCESS-P;
```

Entity Definition for Entity-Type DATA-AREA:

```
Add entity EMPLOYEE(P) entity-type = DATA-AREA-P;

Add entity EMPLOYEE-PAYMENTS(P) entity-type = DATA-AREA-P;

Add entity BENEFITS(P) entity-type = DATA-AREA-P;

Add entity CUSTOMER(P) entity-type = DATA-AREA-P;

Add entity SUPPLIER(P) entity-type = DATA-AREA-P;
```

```
Add entity PRODUCT(P) entity-type = DATA-AREA-P;

Add entity PRODUCT-MARKET(P) entity-type = DATA-AREA-P;

Add entity DEPARTMENT(P) entity-type = DATA-AREA-P;
```

Entity Definition for Entity-Type DATA-SET:

```
Add entity PERSONAL-INFORMATION(P) entity-type =
DATA-SET-P;

Add entity POSITION-DESCRIPTION(P) entity-type =
DATA-SET-P;

Add entity EDUCATION(P) entity-type = DATA-SET-P;

Add entity PERFORMANCE(P) entity-type = DATA-SET-P;

Add entity PAY-CATEGORY(P) entity-type = DATA-SET-P;

Add entity AWARD-CATEGORY(P) entity-type = DATA-SET-P;

Add entity BONUS-CATEGORY(P) entity-type = DATA-SET-P;
```

Relationship Definition between COMPANY and CORPORATION:

```
Add relationship COMPANY-X(P)  COMPANY-IS-PART-OF-
CORPORATION  CORPORATION-XYZ(P);

Add relationship COMPANY-Y(P)  COMPANY-IS-PART-OF-
CORPORATION  CORPORATION-XYZ(P);

Add relationship COMPANY-Z(P)  COMPANY-IS-PART-OF-
CORPORATION  CORPORATION-XYZ(P);
```

Relationship Definition between COMPANY and DIVISION:

```
Add relationship COMPANY-X(P)  COMPANY-CONTAINS-DIVISION
PERSONNEL-BENEFITS-DIVISION(P);

Add relationship COMPANY-X(P)  COMPANY-CONTAINS-DIVISION
ACCOUNTING-DIVISION(P);

Add relationship COMPANY-X(P)  COMPANY-CONTAINS-DIVISION
PRODUCT-DESIGN-DIVISION(P);
```

```
Add relationship COMPANY-X(P)  COMPANY-CONTAINS-DIVISION
MARKETING-ADVERTISING-DIVISION(P);
```

Relationship Definition between DIVISION and DEPT:

```
Add relationship ACCOUNTING-DIVISION(P)
DIVISION-CONTAINS-DEPT  ACCOUNTS-RECEIVABLE-DEPT(P);

Add relationship ACCOUNTING-DIVISION(P)
DIVISION-CONTAINS-DEPT  ACCOUNTS-PAYABLE-DEPT(P);

Add relationship ACCOUNTING-DIVISION(P)
DIVISION-CONTAINS-DEPT  PAYROLL-DEPT(P);

Add relationship ACCOUNTING-DIVISION(P)
DIVISION-CONTAINS-DEPT  COST-ACCOUNTING-DEPT(P);
```

Relationship Definition between SUCCESS-FACTOR and FUNCTIONAL-AREA:

```
Add relationship EFFICIENT-PRODUCTION(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA  PERSONNEL(P);

Add relationship EFFICIENT-PRODUCTION(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA  ACCOUNTING(P);

Add relationship ADVERTISING-AND-DISTRIBUTION(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA   PRODUCT-
MARKETING(P);

Add relationship SKILLED-WORKERS(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA  ACCOUNTING(P);

Add relationship SKILLED-WORKERS(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA  PERSONNEL(P);

Add relationship PRODUCT-INNOVATION(P)
SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
PRODUCT-PLANNING(P);
```

Relationship Definition between FUNCTIONAL-AREA and PROCESS:

```
Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  ACCOUNTS-RECEIVABLE(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-
```

```
PROCESS  ACCOUNTS-PAYABLE(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  EMPLOYEE-PAYROLL(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  COST-ACCOUNTING(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  PERSONNEL-PLANNING(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  RECRUITING(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  EMPLOYEE-TRAINING(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  EMPLOYEE-COMPENSATION(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  EMPLOYEE-BENEFITS(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-
PROCESS  EMPLOYEE-INCENTIVES(P);
```

Relationship Definition between DATA-AREA and DATA-SET:

```
Add relationship EMPLOYEE(P)  DATA-AREA-CONTAINS-DATA-SET
PERSONAL-INFORMATION(P);

Add relationship EMPLOYEE(P)  DATA-AREA-CONTAINS-DATA-SET
POSITION-DESCRIPTION(P);

Add relationship EMPLOYEE(P)  DATA-AREA-CONTAINS-DATA-SET
EDUCATION(P);

Add relationship EMPLOYEE(P)  DATA-AREA-CONTAINS-DATA-SET
PERFORMANCE(P);

Add relationship EMPLOYEE-PAYMENTS(P)  DATA-AREA-CONTAINS-
DATA-SET  PAY-CATEGORY(P);

Add relationship EMPLOYEE-PAYMENTS(P)  DATA-AREA-CONTAINS-
DATA-SET  AWARD-CATEGORY(P);

Add relationship EMPLOYEE-PAYMENTS(P)  DATA-AREA-CONTAINS-
DATA-SET  BONUS-CATEGORY(P);
```

Relationship Definition between FUNCTIONAL-AREA and DATA-AREA:

```
Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  EMPLOYEE(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  EMPLOYEE-PAYMENTS(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  BENEFITS(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  DEPARTMENT(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  CUSTOMER(P);

Add relationship ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  SUPPLIER(P);

Add relationship PERSONNEL(P)   FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA EMPLOYEE(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  EMPLOYEE-PAYMENTS(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  BENEFITS(P);

Add relationship PERSONNEL(P)  FUNCTIONAL-AREA-REFERS-TO-
DATA-AREA  DEPARTMENT(P);

Add relationship PRODUCT-PLANNING(P)  FUNCTIONAL-AREA-
REFERS-TO-DATA-AREA  PRODUCT(P);

Add relationship PRODUCT-PLANNING(P)  FUNCTIONAL-AREA-
REFERS-TO-DATA-AREA  PRODUCT-MARKET(P);
```

Relationship Definition between PROCESS and DEPT:

```
Add relationship EMPLOYEE-PAYROLL(P)  PROCESS-IS-PERFORMED-
BY-DEPT  PAYROLL-DEPT(P);
```

8.6 Results of Strategic Systems Planning

The examples of Strategic Systems Planning show how an organizational entity can capture information in an IRD about success factors, organizational restructuring, functional decomposition, cross-referencing of objectives and functions, data decomposition, and cross-referencing of functions and data.

Management's analysis of the **critical success factors** uncovered a number of problems in functional performance in Company X. While Company X's production efficiency was acceptable, management was not receiving complete information on the total cost to produce each product. This information was provided through a number of organizational units, but no one division or department had the responsibility to provide management with good cost accounting information.

Due to the resulting poor cost accounting information, production efficiency was not being effectively analyzed. When this deficiency was uncovered, Company X management decided to translate the **Cost Accounting** process into a new department within the Accounting Division.

Another problem discovered at Company X was the relatively large turnover in skilled employees. Management had known about the high turnover before, but had not recognized the extent of the problem or examined its causes. A significant number of new employees were receiving training at Company X, working at Company X for several years to gain experience, and then going to work for competitor companies.

This loss of experienced personnel was expensive. As a result, Company X was spending more to recruit, relocate, and train employees. The large percentage of **employee turnover** also affected production efficiency and product quality to some degree, since the skills and experience of the absent personnel would not be used to improve the products or efficiency of Company X. By investigating this problem, Company X found that competitor companies offered **employee incentives** to reward skilled and motivated employees who performed particularly well. Competitor companies also offered better employee **benefits** packages that encouraged the retention of skilled employees.

As a result of this analysis, Company X decided to start its own **Employee Incentives** program and to boost its lagging **Employee Benefits** program with an improved retirement plan. While these program changes would cost more initially, management decided that they would eventually save the company money by reducing employee turnover and giving Company X a greater base of skilled employees. To support these new personnel programs, the management of Company X has assigned a high-priority to the implementation of a new personnel system.

The investigation into employee turnover revealed that employee retention and morale were also affected by **missing and misdirected paychecks.** Due to the company's outdated payroll system, employees often received paychecks for incorrect amounts, or received their paychecks late when paychecks were sent to the wrong employees. These payroll problems gave employees the impression that Company X was unreliable and untrustworthy. While Company X had previously known the problem existed, management had not realized the extent or the affects of the problem. As a result, Company X has now assigned a high-priority to the development of a new payroll system.

As one result of Strategic Systems Planning, the XYZ Corporation has decided to **combine** the data processing and data files used in Company X's Personnel and Accounting functions into one system. Because the functional areas of ACCOUNTING and PERSONNEL both refer to the data areas of EMPLOYEE, EMPLOYEE-PAYMENTS, BENEFITS, and EMPLOYEE-INCENTIVES, these shared data areas are considered to be a good combination for a database. The planned database to support this data will be called the **Accounting and Personnel Database.** This database will be coordinated with a number of data processing, data conversion, and data communications programs in the new **Accounting and Personnel System**, which will include the new payroll system.

The influence of the critical success factors defined by high-level management give this project a high-priority for system development. Company X wants to show rapid improvement in performing these functions, since these functions are influenced by **success factors.** The Accounting and Personnel System has been scheduled for system development and a team is now being assembled for the Requirements Definition phase.

9.0 Conclusions

This guide describes the Information Resource Dictionary System (IRDS) and its applications, Information Resource Dictionaries (IRDs). Metadata to be stored in an IRD is differentiated from data to be stored in a database. The status of IRDS standardization is discussed.

Due to the enormous volume of interrelated information that must be handled during system development, a data dictionary system such as the IRDS should be used to support the system life cycle. The IRDS standard is particularly recommended for information system life cycle support due to the comprehensive facilities the IRDS provides, which many other data dictionary systems do not.

The extensible schema capability of the IRDS permits the user to customize one or more Entity-Relationship-Attribute models for any application. The Life Cycle Phase Facility gives the IRDS user the capability to support information for multiple, cross-referenced life cycle phases that are accessed through views. The IRD-IRD Interface allows the user to transfer information from one IRD to another, for a variety of purposes such as information consolidation or archives.

The IRDS should be used to support Information Resource Management (IRM) and Data Administration. IRDS applications for Data Administration, such as Data Element Standardization, are described. The development of the IRDS is illustrated in terms of the evolution from Data Processing to IRM. Features of the IRDS standard are discussed in terms of both functionality and applications development. Procedures and commands for IRD application development are defined.

This guide illustrates the use of the IRDS during the first system development life cycle phase, Strategic Systems Planning, through the definition of:

- Example problem statements representative of the information to be supported during the Strategic Systems Planning phase.
- An Entity-Relationship-Attribute model that represents each problem statement.
- Schema definition commands that represent the Entity-Relationship-Attribute model in an Information Resource Dictionary.
- Metadata definition commands that represent the information to be collected about the target system.

APPENDIX:
Extract of IRD Output for Strategic Systems Planning

```
------------------------------------------------------------

Entity = COMPANY-X(P)
Entity Type = COMPANY-P

                  Attributes
ADDED-BY = law
LOCATION = CHICAGO
NUMBER-OF-TIMES-MODIFIED = 0

               Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 172324

                  Relationships
COMPANY-X(P) COMPANY-CONTAINS-DIVISION ACCOUNTING-DIVISION(P)
COMPANY-X(P)  COMPANY-CONTAINS-DIVISION  MARKETING-ANALYSIS-
     DIVISION(P)
COMPANY-X(P)  COMPANY-CONTAINS-DIVISION  PERSONNEL-BENEFITS-
     DIVISION(P)
COMPANY-X(P) COMPANY-CONTAINS-DIVISION PRODUCT-DESIGN-DIVISION(P)
COMPANY-X(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)

------------------------------------------------------------

Entity = COMPANY-Y(P)
Entity Type = COMPANY-P

                  Attributes
ADDED-BY = law
LOCATION = ST. LOUIS
NUMBER-OF-TIMES-MODIFIED = 0

               Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 172543

                  Relationships
COMPANY-Y(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)

------------------------------------------------------------
```

```
Entity = COMPANY-Z(P)
Entity Type = COMPANY-P

                  Attributes
ADDED-BY = law
LOCATION = CINCINNATI
NUMBER-OF-TIMES-MODIFIED = 0

               Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 172754

                  Relationships
COMPANY-Z(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)

-----------------------------------------------------------------

Entity = CORPORATION-XYZ(P)
Entity Type = CORPORATION-P

                  Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0

               Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 172203

                Relationships
COMPANY-X(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)
COMPANY-Y(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)
COMPANY-Z(P) COMPANY-IS-PART-OF-CORPORATION CORPORATION-XYZ(P)

-----------------------------------------------------------------

Entity = EMPLOYEE(P)
Entity Type = DATA-AREA-P

                  Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0

               Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 181984200
```

```
                Relationships
EMPLOYEE(P) DATA-AREA-CONTAINS-DATA-SET EDUCATION(P)
EMPLOYEE(P) DATA-AREA-CONTAINS-DATA-SET PERFORMANCE(P)
EMPLOYEE(P) DATA-AREA-CONTAINS-DATA-SET PERSONAL-INFORMATION(P)
EMPLOYEE(P) DATA-AREA-CONTAINS-DATA-SET POSITION-DESCRIPTION(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA EMPLOYEE(P)
PERSONNEL(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA EMPLOYEE(P)

-----------------------------------------------------------------

Entity = EMPLOYEE-PAYMENTS(P)
Entity Type = DATA-AREA-P

                Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0

             Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 184427

                Relationships
EMPLOYEE-PAYMENTS(P) DATA-AREA-CONTAINS-DATA-SET AWARD-
CATEGORY(P)
EMPLOYEE-PAYMENTS(P) DATA-AREA-CONTAINS-DATA-SET BONUS-
CATEGORY(P)
EMPLOYEE-PAYMENTS(P) DATA-AREA-CONTAINS-DATA-SET PAY-CATEGORY(P)
ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-DATA-AREA  EMPLOYEE-
     PAYMENTS(P)
PERSONNEL(P)  FUNCTIONAL-AREA-REFERS-TO-DATA-AREA  EMPLOYEE-
     PAYMENTS(P)

-----------------------------------------------------------------

Entity = ACCOUNTING-DIVISION(P)
Entity Type = DIVISION-P

                Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0



             Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 173255
```

```
                Relationships
COMPANY-X(P) COMPANY-CONTAINS-DIVISION ACCOUNTING-DIVISION(P)
ACCOUNTING-DIVISION(P)  DIVISION-CONTAINS-DEPT  ACCOUNTS-PAYABLE-
     DEPT(P)
ACCOUNTING-DIVISION(P)  DIVISION-CONTAINS-DEPT  ACCOUNTS-
RECEIVABLE-DEPT(P)
ACCOUNTING-DIVISION(P)  DIVISION-CONTAINS-DEPT  COST-ACCOUNTING-
     DEPT(P)
ACCOUNTING-DIVISION(P) DIVISION-CONTAINS-DEPT PAYROLL-DEPT(P)

-------------------------------------------------------------

Entity = ACCOUNTING(P)
Entity Type = FUNCTIONAL-AREA-P

                Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0

             Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 180237

                Relationships
ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  ACCOUNTS-
PAYABLE(P)
ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  ACCOUNTS-
RECEIVABLE(P)
ACCOUNTING(P) FUNCTIONAL-AREA-CONTAINS-PROCESS COST-ACCOUNTING(P)
ACCOUNTING(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  EMPLOYEE-
PAYROLL(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA BENEFITS(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA CUSTOMER(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA DEPARTMENT(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA EMPLOYEE(P)
ACCOUNTING(P)  FUNCTIONAL-AREA-REFERS-TO-DATA-AREA  EMPLOYEE-
     PAYMENTS(P)
ACCOUNTING(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA SUPPLIER(P)
EFFICIENT-PRODUCTION(P) SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
     ACCOUNTING(P)
SKILLED-WORKERS(P)  SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
     ACCOUNTING(P)

-------------------------------------------------------------

Entity = PERSONNEL(P)
Entity Type = FUNCTIONAL-AREA-P

                Attributes
ADDED-BY = law
NUMBER-OF-TIMES-MODIFIED = 0
```

```
              Attribute Groups
     DATE-TIME-ADDED
SYSTEM-DATE = 19871210
SYSTEM-TIME = 180520

                Relationships
PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  EMPLOYEE-
BENEFITS(P)
PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  EMPLOYEE-
COMPENSATION(P)
PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  EMPLOYEE-
INCENTIVES(P)
PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  EMPLOYEE-
TRAINING(P)
PERSONNEL(P)  FUNCTIONAL-AREA-CONTAINS-PROCESS  PERSONNEL-
PLANNING(P)
PERSONNEL(P) FUNCTIONAL-AREA-CONTAINS-PROCESS RECRUITING(P)
PERSONNEL(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA BENEFITS(P)
PERSONNEL(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA DEPARTMENT(P)
PERSONNEL(P) FUNCTIONAL-AREA-REFERS-TO-DATA-AREA EMPLOYEE(P)
PERSONNEL(P)  FUNCTIONAL-AREA-REFERS-TO-DATA-AREA  EMPLOYEE-
     PAYMENTS(P)
EFFICIENT-PRODUCTION(P) SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
     PERSONNEL(P)
SKILLED-WORKERS(P)  SUCCESS-FACTOR-INFLUENCES-FUNCTIONAL-AREA
     PERSONNEL(P)

-----------------------------------------------------------------
```

GLOSSARY

Access name - In an IRD at schema or metadata level, the primary identifier of each entity or meta-entity; the name by which the entity is known to the user; an access name consists of an assigned access name and version identifier.

Assigned access name - In an IRD at metadata level, a user-assigned or system-assigned name which provides unique access to an entity when first added to the IRD; in an IRD at schema level, a user-assigned name which provides unique access to a meta-entity when first added to the IRD.

Assigned descriptive name - In an IRD at metadata level, a name for an entity or meta-entity which is more descriptive than the assigned name.

Attribute - In an IRD at metadata level, a property that describes, quantifies, or defines the representation of an entity or relationship.

Attribute-group - In an IRD at metadata level, an ordered set of two or more attributes used together (e.g., the attribute-group of DATE-TIME is made up of the attributes DATE and TIME).

Attribute-group-type - In an IRD at schema level, an ordered set of two or more attributes used together (e.g., the attribute-group-type DATE-TIME-ADDED is made up of attribute-types SYSTEM-TIME and SYSTEM-DATE).

Attribute-type - In an IRD at schema level, the label for a set of attributes which may be common to an entity-type or relationship-type.

Data Administration - The responsibility for definition, organization, supervision, and protection of data within an enterprise or organization.

Data Administrator - A person or group that ensures the utility of data used within an organization, by defining data policies and standards, planning for the efficient use of data, coordinating data structures among organizational components, performing logical database design, and defining data security procedures.

Database Administrator - A person or group which provides technical support for one or more databases, by defining database schemas and subschemas, maintaining data integrity and concurrency, providing physical database design for performance optimization, and enforcing the policies, standards, and procedures set by the Data Administrator.

Data dictionary - A specialized type of database containing metadata that is managed by a data dictionary system; a repository of information describing the characteristics of data used to design, monitor, document, protect, and control data in information systems and databases; an application of a data dictionary system.

Data dictionary system - An automated system such as an IRDS that can support one or more data dictionaries.

Data integrity - In information processing, the condition in which data is accurate, current, consistent, and complete.

Data security - The protection of data from accidental or malicious modification or destruction, and from accidental or intentional disclosure to unauthorized personnel.

Data structure - The logical relationships which exist among units of data and the descriptive features defined for those relationships and data units; an instance or occurrence of a data model.

Database Management System (DBMS) - A software system that provides the functionality to support the creation, access, maintenance, and control of databases, and that facilitates the execution of application programs using data from these databases.

Descriptive name - In an IRD at metadata level, a unique and more descriptive name for the access name; consists of assigned descriptive name and version identifier.

Entity - In an IRD at metadata level, any person, place, thing, concept, or event about which metadata can be collected.

Entity-Relationship Model (E-R Model) - An information model based on the concept of entities, relationships among entities, and attributes of entities and relationships; also known as Entity-Relationship-Attribute Model (E-R-A Model).

Entity-type - In an IRD at schema level, the label for a set of entities which have a similar concept and share a set of common attribute-types.

Export/import - In an IRDS, pertains to that set of commands, controls, and other procedural elements necessary to move the contents of one IRD to another. (See Portability.)

Functionality - The capability of a software system to perform a function.

Global data model - A high-level, top-down description of an enterprise's data in a manner that reflects the information structure of the entire enterprise.

Information Resource Dictionary (IRD) - A data dictionary application managed by an IRDS; a collection of entities, relationships, and attributes used by an organization to model its information environment.

Information Resource Dictionary System (IRDS) - A set of standard specifications for a data dictionary system resulting from U.S. Federal and national standards efforts; a computer software system conforming to these standards that provides facilities for recording, storing, and processing descriptions of an organization's significant information and information processing resources.

Information Resource Management (IRM) - The responsibility for planning, organizing, and controlling information for coordinated use in data management, data processing, data communications, and data conversion, in a manner consistent with the primary goals and objectives of the enterprise.

Interface - A point of communication between two or more persons, processes, software systems, or other physical entities.

Interoperability - In information processing, a characteristic of software that allows it to be run on more than one type or size of computer and under more than one operating system.

IRD-IRD interface - The means of transporting all or part of an IRD's schema and metadata to another IRD, and of comparing the schema of one IRD to the schema of another to establish compatibility between IRDs. (See Export/import and Portability.)

IRD schema - A model of the logical structure of the IRD, consisting of components such as entity-types, relationship-types, and attribute-types.

IRD schema descriptor - Any meta-entity in the IRD schema, such as an entity-type, attribute-type, or relationship-type.

IRD schema extensibility - The capability to add new IRD schema descriptors (i.e., new entity-types, attribute-types, and attribute-types) to any IRD schema.

Life cycle phase - A primary activity, or phase, in the complete life cycle of an information system; in an IRDS, a phase in the life of an IRD entity, or set of IRD entities, used as a basis for a logical partition of an IRD; a logical partition of an IRD that is used to represent information about a phase in the life cycle of an information system.

Meta-attribute - In an IRD at schema level, a property of a meta-entity or meta-relationship of its schema

Meta-attribute-group - In an IRD, at schema level, an ordered set of two or more meta-attributes used together.

Metadata - Information describing the characteristics of data; data or information about data; descriptive information about an organization's data, data activities, systems, and holdings.

Metadata model - A collection of Entity-Relationship-Attribute models, used to describe the schema requirements for an IRD, and the metadata the E-R-A models support, used to populate that IRD.

Meta-entity - In an IRD at schema level, a schema entity for entity-types, relationship-types, and attribute-types.

Meta-relationship - In an IRD at schema level, a schema relationship between schema entities (i.e., between meta-entities).

Meta-relationship-class-type - In an IRD at schema level, a set of meta-relationships that contain the same verb (e.g., [meta-entity-type-1]-VERB-[meta-entity-type-2]).

Panel interface - In an IRDS, a screen-oriented user interface designed to provide users with information about an IRD application and permit easier interactive processing.

Partition - A boundary maintained within the logical storage area used by a software system, such as an IRDS, that permits separate areas to be allocated for different purposes, and to which different priorities and access permissions may apply.

Portability - Transportability; in information processing, the ability to transfer data or metadata from one system to another without being required to recreate or reenter data descriptions or to significantly modify the application being transported. (See Interoperability.)

Relationship - In an IRD at metadata level, a directed association between entities.

Relationship-class - In an IRD at metadata level, the common verb shared by a set of relationships (e.g., CONTAINS).

Relationship-class-type - In an IRD at schema level, a set of relationship-types which use the same verb in the central position (e.g., [entity-type-1]-VERB-[entity-type-2]);

Relationship-type - In an IRD at schema level, the label for a set of relationships which have similar meanings and share a set of common attribute-types.

Revision number - In an IRD at schema or metadata level, a positive integer consecutively assigned to each change affecting an IRD meta-entity or entity, respectively; part of the version identifier of the assigned access name. (See Version identifier.)

System life cycle - Those phases and activities associated with, for example, the analysis, specification, design, development, testing, integration, operation, maintenance and modification of a software system. (See Life cycle phase.)

Transportability - In information processing, the ability to transfer data or metadata from one system to another without being required to recreate or reenter data descriptions or to significantly modify the application being transported. (See Interoperability.)

User - In information processing, an individual, organization, or facility that makes use of an information system or other software system, such as an IRDS.

Validation - In information processing, the checking of data for correctness or compliance with applicable standards, rules, and conventions.

Variation name - In an IRD at schema or metadata level, a label which identifies each entity or meta-entity, respectively, with the same assigned access or descriptive name; part of the version identifier of the assigned access name. (See Version identifier.)

Version identifier - Last part of the assigned access name of an entity or meta-entity; composed of the variation name and the revision number of the entity or meta-entity. (See Variation name and Revision number.)

View - In an IRD, a set of specified entity-types and relationship-types within a single life cycle phase through which users define and access metadata; a view may be associated with one or more IRD users and, conversely, an IRD user may be associated with one or more views.

REFERENCES

[ANSI86] ANSI, Draft Proposed American National Standard IRDS, Document ISO/TC97/SC21/WG3 N166R1, American National Standards Institute, 1430 Broadway, New York, NY, November 1986. See also [X3H487].

[CHEN79] Chen, Peter P., editor, Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design, held December 10-12, 1979, North-Holland Publishing Co., Amsterdam, The Netherlands, 1980.

[CROW84] Crowdley, Ellen T., editor, Acronyms, Initialisms, and Abbreviations, Gale Research Co., Detroit, MI, 1984.

[DATE86] Date, Chris J., An Introduction to Database Systems Vol.I, Addison-Wesley Publishing Co., Fourth Edition, Reading, MA, 1986.

[DATE83] Date, Chris J., An Introduction to Database Systems Vol.II, Addison-Wesley Publishing Co., Reading, MA,1986.

[DEMA79] DeMarco, Tom, Structured Analysis and System Specification, Yourdon Press, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1979.

[DOLL78] Doll, Dixon R., Data Communications: Facilities, Networks, and Systems Design, John Wiley & Sons, New York, NY, 1978.

[DURE85] Durell, William R., Data Administration: A Practical Guide to Successful Data Management, McGraw-Hill Book Co., New York, NY, 1985.

[FISH87] Fisher, Gary E., Application Software Prototyping and Fourth Generation Languages, NBS Special Publication 500-148, National Bureau of Standards, Gaithersburg, MD, May 1987.

[FLAV81] Flavin, Matt, Fundamental Concepts of Information Modeling, Yourdon Press, Inc., New York, NY, 1981.

[FONG86] Fong, Elizabeth N., and Alan H. Goldfine, editors, Information Resource Management - Making It Work, NBS Special Publication 500-139, National Bureau of Standards, Gaithersburg, MD, June 1986.

[FONG85] Fong, Elizabeth N., Margaret W. Henderson [Law], David K. Jefferson, and Joan M. Sullivan, Guide on Logical Database Design, NBS Special Publication 500-122, National Bureau of Standards, Gaithersburg, MD, February 1985.

[GALL82] Frank J. Galland, editor, Dictionary of Computing, John Wiley & Sons, New York, 1982.

[GOLD88a] Goldfine, Alan, and Patricia Konig, A Technical Overview of the Information Resource Dictionary System (Second Edition), NBSIR 88-3700, National Bureau of Standards, Gaithersburg, MD, January 1988.

[GOLD88b] Goldfine, Alan, Using the Information Resource Dictionary System Command Language (Second Edition), NBSIR 88-3701, National Bureau of Standards, Gaithersburg, MD, January 1988.

[GOLD82] Goldfine, Alan H., editor, Data Base Directions: Information Resource Management--Strategies and Tools, NBS Special Publication 500-92, National Bureau of Standards, Gaithersburg, MD, September 1982.

[HENI80] Heninger, Kathryn L., "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," IEEE Transactions on Software Engineering, Vol. SE-6, No. 1, January 1980.

[ISO82] ISO TC97/SC5/WG3 Conceptual Schema, Appendix D: The Entity-Attribute-Relationship Approaches, ISO publication, March 1982.

[LEON82] Leong-Hong, Belkis W., and Bernard K. Plagman, Data Dictionary/Directory Systems: Administration, Implementation and Usage, John Wiley & Sons, New York, NY, 1982.

[LEFK83] Lefkovits, Henry C., Edgar H. Sibley, and Sandra L. Lefkovits, Information Resource/Data Dictionary Systems, QED Information Sciences, Inc., Wellesley, MA, 1983.

[LOOM86] Loomis, Mary E. S., "Data Modeling -- The IDEF1X Technique," Proceedings of the 1986 IEEE Conference on Computers and Communications, IEEE Computer Society, IEEE Service Center, Piscataway, NJ, 1986.

[LOOM85] Loomis, Mary E. S., "Logical Data Modeling -- A Step Toward Integration," Proceedings of Conference on Computer-Aided Technologies COMPINT 1985, held in Montreal, Quebec, Canada, IEEE Computer Society, IEEE Service Center, Piscataway, NJ, 1986.

[MART82] Martin, James, Strategic Data-Planning Methodologies, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[MAYN84] Mayne, Alan, The Uses of a Dictionary System, The National Computing Centre Limited, Manchester, England, 1984.

[NEWM82] Newman, David T., editor, Information Resources Management: Conference Proceedings 1982, National Institute for Management Research, Santa Monica, CA, February 1982.

[NEWT87] Newton, Judith J., Guide on Data Entity Naming Conventions, NBS Special Publication 500-149, National Bureau of Standards, Gaithersburg, MD, October 1987.

[PAGE80] Page-Jones, Meiler, The Practical Guide to Structured Systems Design, Yourdon Press, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.

[PERK85] Perkinson, Richard C., Data Analysis: The Key to Data Base Design, QED Information Sciences, Inc., Wellesley, MA, 1985.

[TEOR86] Teorey, Toby J, Dongqing Yang, and James P. Fry, "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," ACM Computing Surveys, Vol. 18, No. 2, June 1986, pp. 197-222.

[VAND82] Van Duyn, Julia, Developing a Data Dictionary System, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[WERT86] Wertz, Charles J., The Data Dictionary: Concepts and Uses, QED Information Sciences, Inc., Wellesley, MA, 1986.

[X3H487] X3H4, "U.S. Member Body Proposed Changes to [ANSI document] N166R1," Accredited Standards Committee X3 for Information Processing Systems, X3 Secretariat, Computer and Business Equipment Manufacturers Association, Washington, DC, March 1987.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** (See instructions)	1. PUBLICATION OR REPORT NO. NBS SP 500/152	2. Performing Organ. Report No.	3. Publication Date April 1988

4. TITLE AND SUBTITLE

Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning

5. AUTHOR(S)

Margaret Henderson Law

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Same as item #6 above.

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 88-600529

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The guide describes the Information Resource Dictionary System (IRDS) and its applications, Information Resource Dictionaries (IRDs). Metadata to be stored in an IRD is differentiated from data to be stored in a database. The role of the IRDS in Information Resource Management (IRM) and Data Administration is discussed. The development of the IRDS is described in terms of the evolution of data processing toward larger, more complex systems that require greater control. With examples drawn from the first phase of the life cycle, the Strategic Systems Planning phase, the guide demonstrates how: (1) to develop example problem statements indicative of the information to be represented in an IRD; (2) to conceive of and define phase partitions and views through which to access information in an IRD; (3) to develop Entity-Relationship-Attribute models for each example problem statement; and (4) to use the IRDS command language in defining an IRD schema and in populating the IRD with metadata. Procedures for using the IRDS extensible schema capability are illustrated.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Data Administration; data dictionary system; data management, data modeling; Entity-Relationship model; E-R; Federal Information Processing Standard; FIPS; Information Resource Dictionary System; Information Resource Management; IRDS; Strategic Systems Planning

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

145

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

☆ U.S. GOVERNMENT PRINTING OFFICE: 1988— 2 0 1 - 5 9 7 / 8 2 5 6 6

NBS Technical Publications

Periodical

Journal of Research—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).
NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.
Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*
Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Bureau of Standards
Gaithersburg, MD 20899

Stimulating America s Progress
1913-1988